**AFRL-SN-RS-TR-2003-272**
**Final Technical Report**
**November 2003**


# EMBEDDED HIGH PERFORMANCE SCALABLE COMPUTING SYSTEMS

**Sanders, A Lockheed Martin Company**

**Sponsored by**
**Defense Advanced Research Projects Agency**
**DARPA Order No. C201**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*



**AIR FORCE RESEARCH LABORATORY**
**SENSORS DIRECTORATE**
**ROME RESEARCH SITE**
**ROME, NEW YORK**

**STINFO FINAL REPORT**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-SN-RS-TR-2003-272 has been reviewed and is approved for publication.

APPROVED: /s/
RALPH KOHLER
Project Engineer

FOR THE DIRECTOR: /s/
RICHARD G. SHAUGHNESSY, Lt Col, USAF
Chief, Rome Operations Office
Sensors Directorate

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | November 2003 | Final Feb 95 – Apr 98 |

**4. TITLE AND SUBTITLE**

EMBEDDED HIGH PEFORMANCE SCALABLE COMPUTING SYSTEMS

**5. FUNDING NUMBERS**

C - F30602-95-2-0013
PE - 62301E
PR - C201
TA - 01
WU - P1

**6. AUTHOR(S)**

David Ngo

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Sanders, A Lockheed Martin Company
PO Box 868
Nashua, NH 03061-0868

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

DARPA
3701 North Fairfax Drive
Arlington, VA 22203-1714

AFRL/SNRT
26 Electronic Pky
Rome, NY 13441-4514

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

AFRL-SN-RS-TR-2003-272

**11. SUPPLEMENTARY NOTES**

AFRL Project Engineer: Ralph Kohler, SNRT, 315-330-2018, kohlerr@rl.af.mil

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**

Approved for public release; distribution unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** ***(Maximum 200 Words)***

The Embedded High Performance Scalable Computing Systems (EHPSCS) program is a cooperative agreement between Sanders, A Lockheed Martin Company and DARPA that ran for three years, from Apr 95 – Apr 98. The focus of the EHPSCS research program was on the development of a highly integrated, scalable multiprocessing architecture based on leading COTS technologies for environmentally constrained applications. The program developed an 11 GFLOPS embedded processor hardware/software testbed and software development tools to facilitate technology transfer, an advanced packaging insertion approach, and a second generation microarchitecture – the ReConfigurable Transport Engine.

**14. SUBJECT TERMS**

Automatic Target Recognition, Synthetic Aperture Radar, Digital Signal Processing (DSP), Two-Level Multicomputer Architecture, Myrinet LAN protocol, Message Passing Interface (MPI)

**15. NUMBER OF PAGES**

80

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

# Table of Contents

# Table of Figures

## List of Tables

## Abstract

The Embedded High Performance Scalable Computing Systems (EHPSCS) program is a contractual cooperative agreement between Sanders, a Lockheed Martin Company and the Defense Advanced Research Projects Agency (DARPA) that ran for three years, from April 1995 to April 1998. The focus of the EHPSCS research program was on the development of a highly integrated, scalable multiprocessing architecture based on leading COTS technologies for environmentally constrained applications. The program developed an 11 GFLOPS embedded processor hardware/software testbed and software development tools to facilitate technology transfer, an advanced packaging insertion approach, and a second generation microarchitecture – the ReConfigurable Transport Engine.

# 1 The Need

The Defense Advanced Research Projects Agency (DARPA) has a defined need to leverage and enhance the commercial scalable High Performance Computing (HPC) technology base for a wide range of embedded military and defense applications. Applications such as Automatic Target Recognition and Synthetic Aperture Radar require tera-ops per second of processing power and gigabytes per second of I/O throughputs. In addition, advanced embedded system developments require a cost-effective environment which provides performance models for critical system analysis and trade decisions, architecture and tools framework to accelerate application development, and embedded resource building blocks to accelerate system implementation. The EHPSCS program has addressed these needs.

# 2 Objective

The main objective of the EHPSCS program was to develop innovative embedded scalable multicomputer solutions based on leading COTS technologies that will support a diverse set of military applications and requirements. These requirements include a wide range of processing power, memory capacity, and resource types that necessitate scalability to address these ranges. Operating environments can range from land to air to space, while volume allocation can range from cubic feet to cubic inches. Lastly, these applications require heterogeneous resource support for flexibility, application-specific requirements, and insertion of next-generation technology to fully maximize size, weight, and performance.

As part of the main objective, the program was to develop a multicomputer hardware/software architecture to meet the wide-ranging requirements and would do so while leveraging as much commercial technology as possible to ease usability and application development and reduce life cycle costs. Finally, this technology was to be demonstrated and transitioned to application programs and to the commercial HPC community.

Specifically, the EHPSC program objectives were to:

- Perform architecture design and analysis of the EHPSCS two-level multicomputer concept.
- Demonstrate architecture development tools.
- Implement a scalable, embedded hardware/software multicomputer functional testbed based on leading COTS technology.
- Demonstrate and benchmark an insertion application algorithm running on the multicomputer functional testbed.
- Investigate and implement advanced packaging concept for embedded application transition of the EHPSCS technologies.
- Develop and demonstrate the ReConfigurable Transport Engine (RCTE) for next-generation application requirements.
- Develop and deliver a COTS-based multicomputer debugger.

# 3 Approach

The Embedded HPSCS research program will combine advanced Digital Signal Processing (DSP), multicomputing, software, and packaging technologies to produce a prototype testbed system for use in a variety of defense and commercial applications that require high processing and I/O bandwidth in environmentally constrained configurations. The testbed will be architected to make use of the latest in commercial technologies to provide scalability via a switched network and an innovative software architecture. Advanced packaging technologies will be investigated to not only address size, weight, and power, but to also address cost and reusability. These technologies will be demonstrated and made available for commercialization as well as to application programs.

## 3.1 EHPSC Two-Level Multicomputer Architecture

To support such wide-ranging objectives, the EHPSCS program has adopted an architecture approach that is based on a two-level multicomputer concept. This architecture must support applications requiring GOPs of aggregate processing throughput and provide real-time network connections across a scalable number of heterogeneous resources. A typical scalable system is shown in Figure 1.

A processing system based on the EHPSCS architecture can be viewed as a single machine that contains multiple functionally cohesive subsystems. The subsystems are networked by a loosely coupled real-time network. Real-time, real-world signals enter and exit the EHPSCS system on the Sensor and Downlink subsystems. The Processor and Memory subsystems are required to execute the algorithms. Each subsystem is in turn composed of a variable number of nodes, and each node has an input and output connection to the network.

Under the two-level architectural concept, the first-level computer, which is the network interface controller (NIC), is separate from the second-level computer, which is an application processor. Each level is implemented with its own complete hardware and software layer that is decoupled from the other. This encapsulation of the real-time network I/O functions permits native resource operating system (OS) and tools support and increased resource efficiency by eliminating the need to handle network traffic. The result is a resource fully dedicated to the execution of the user application which increases the effective throughput efficiency of the multicomputer system.

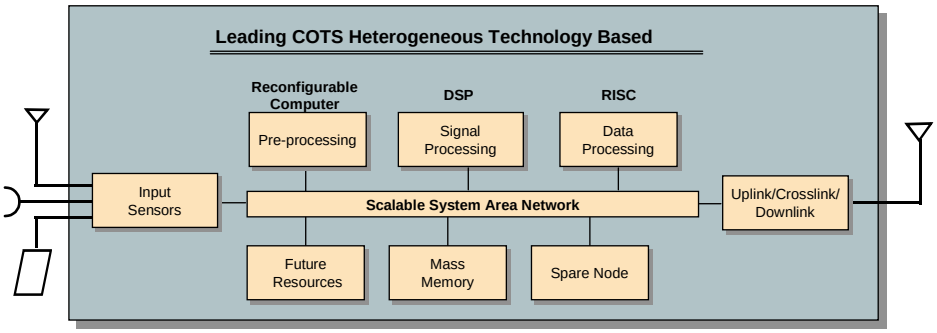



**Figure 1. Embedded High Performance Scalable Computing Architecture Concept**

The application processor is also sometimes referred to as a resource. Note that the term "resource" referred to in this document can be of any system functionality such as a compute resource, a sensor interface, memory, etc. The EHPSCS two-level architecture concept is shown in Figure 2.

A conscious effort was made during the architecture design phase of this program to adopt an open architecture based on COTS components and avoid a proprietary system. This was achieved with the two-level multicomputer concept in concert with a COTS-based, switched network interconnect. Several attributes result from this COTS-based, two-level architecture. It is scalable, meaning that resources can be added or subtracted from the network based on application needs. It is expandable, permitting a heterogeneous system. It is flexible, meaning different applications can run on the same hardware. It is modular, such that each resource node is an independent computer that can be combined to form a flexible system configuration. And lastly, application software is portable allowing development and execution on multiple, different platforms thus preserving the investment made in software development as the hardware evolves or changes. These features will be discussed in more detail in the following sections.

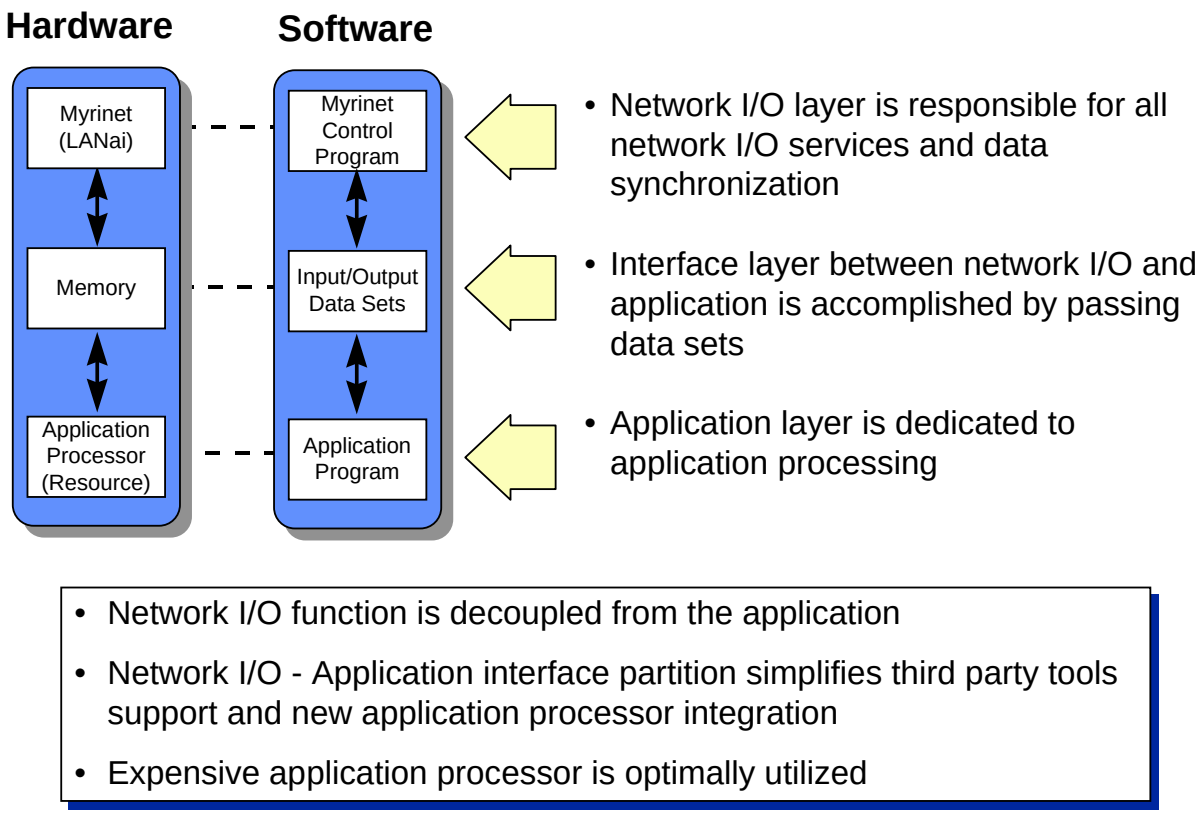



**Figure 2. EHPSCS Two-Level Multicomputer Architecture Concept**

The hardware function in the NIC is based on the LANai network interface chip from Myricom. The LANai is a RISC based NIC chip that provides the network processor and protocol interface to the Myrinet network. The network software function is implemented by a Myrinet Control Program (MCP) which is executed by the LANai. The MCP network software in the EHPSCS architecture is called the Data Synchronization Queue (DSQ). The LANai and DSQ are combined to provide the real-time scalable network solution for embedded system application. The network function is fully isolated from the application layer to offer maximum portability in support of diverse processing resources. The application processor was chosen based on leading COTS DSP microprocessor technology. The DSP compute node is currently based on the 21060, a representative state-of-the-art DSP technology from Analog Devices, Inc.

### 3.2 Myrinet Overview

Myrinet is a high-performance, packet-based switched network that is the preferred network for the EHPSC program. The Myrinet was developed under previous DARPA sponsorship to advance the embedded high-performance computing technology base.

Each Myrinet link is bidirectional and full duplex providing 160 Mbytes/sec in each direction. Low-latency cut-through crossbar switches of up to 16 ports provide scalability across the network. Unlike other networks that share a communication medium, the aggregate throughput of the Myrinet increases as the network scales up in size. The Myrinet switching technology provides scalability and a uniform processor interconnect at multiple levels in the architecture: from board level to backplane to System Area Network (SAN) and Local Area Network (LAN).

Myrinet exists in two protocols – a SAN protocol for board-level and backplane-level communication and a LAN protocol for system-to-system and longer distance communication. Each protocol provides flow control and error control on each port.

Any network topology is allowed. The single-port interfaces and multiple-port switches may be connected by links in any network topology, including networks that provide redundant paths for performance and fault tolerance. The Myrinet interfaces map the network, and use whatever paths are available from host to host. Myricom provides board support software tools for interfacing host platforms with embedded Myrinet topologies and Sanders has built on these tools to provide board support software for the EHPSCS APU.

Myrinet packets may be of any length, and thus can encapsulate other types of packets, including IP packets, without an adaptation layer. Each packet is identified by type so that a Myrinet, like an Ethernet, can carry packets of many types or protocols simultaneously. Thus, a Myrinet supports several software interfaces.

Myrinet building blocks consist of four basic components. These are the LANai, the crossbar switch, the Myrinet Interface (MI) and the FIFO Interface (FI). The LANai and the crossbar switch have been described, previously. The MI chip provides the electrical conversion between the SAN protocol and the LAN protocol. The FI chip is basically the SAN interface subset of the LANai. It interfaces directly to a SAN port and consists of a SAN protocol interface on one end and a parallel FIFO interface on the other.

For more detailed information on the Myrinet network, refer to either Myricom's web site at www.myri.com or the attachment, High Performance Scalable Computing Distributed Architecture Resource Controller Technical Reference, Revision 1.1.

### 3.3 Testbed Development Approach

A major objective of the EHPSCS program is to develop a scalable, embedded hardware/software multicomputer functional testbed based on leading COTS technology.

This functional testbed is the hardware and software realization of the two-level multicomputer architecture concept described in Section 3.1.

The approach taken in the hardware implementation was to leverage as much from leading edge commercial technologies and develop what was necessary to demonstrate the architectural concept and promote future technology insertion. The Analog Devices Super Harvard Architecture (SHARC) 21060/62 processor was used as the core commodity processor for clusters of multiprocessing nodes which will be linked by an embedded Myrinet switching network to enable scalable low latency, high bandwidth interprocessor communication. In addition, a transition path to highly constrained embedded applications was developed and demonstrated with advanced packaging research on the testbed hardware.

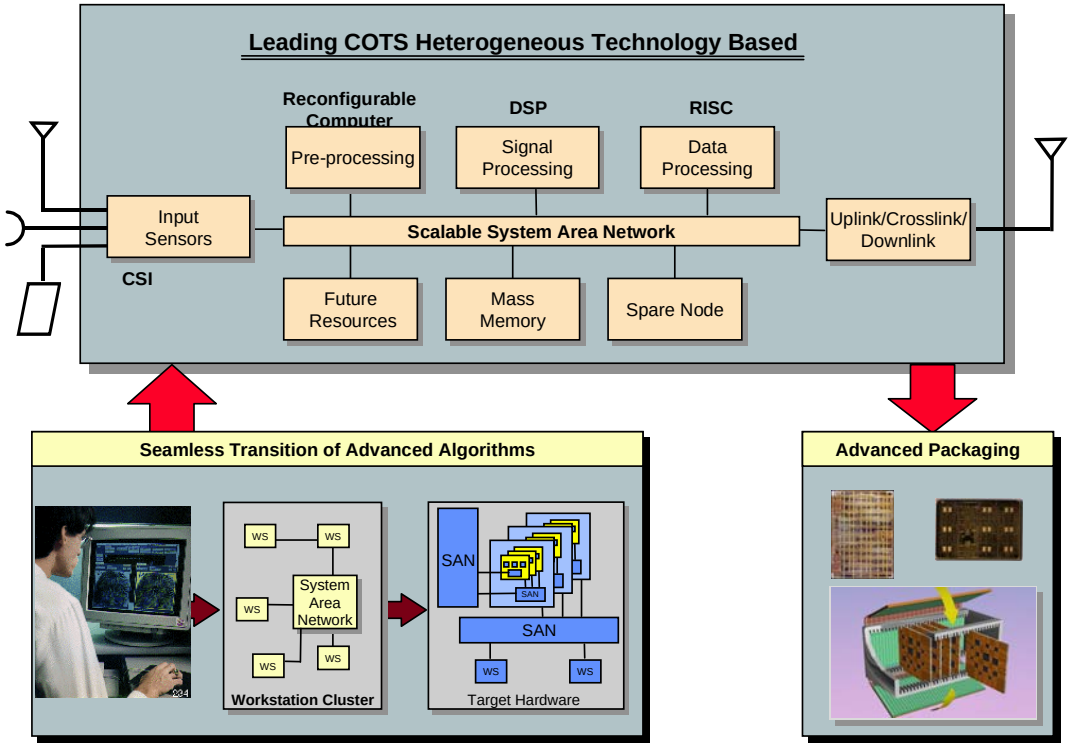



**Figure 3. EHPSCS Multicomputer Architecture Concept**

The underlying software philosophy was to provide a suite of visualization tools to support the development of low latency real-time, complex large scale applications. This philosophy offers the system application developer an environment independent of the underlying resources and network interconnect of the two-level multicomputer. The testbed enables portability, reuse, and target interoperability of advanced signal processing algorithms such that they can be developed in a workstation environment and transitioned seamlessly to real-time systems. The development environment includes tool sets for system architecture analysis and profiling, multiprocessor debugging, and a variety of resource compilers, linkers, and real-time kernels.

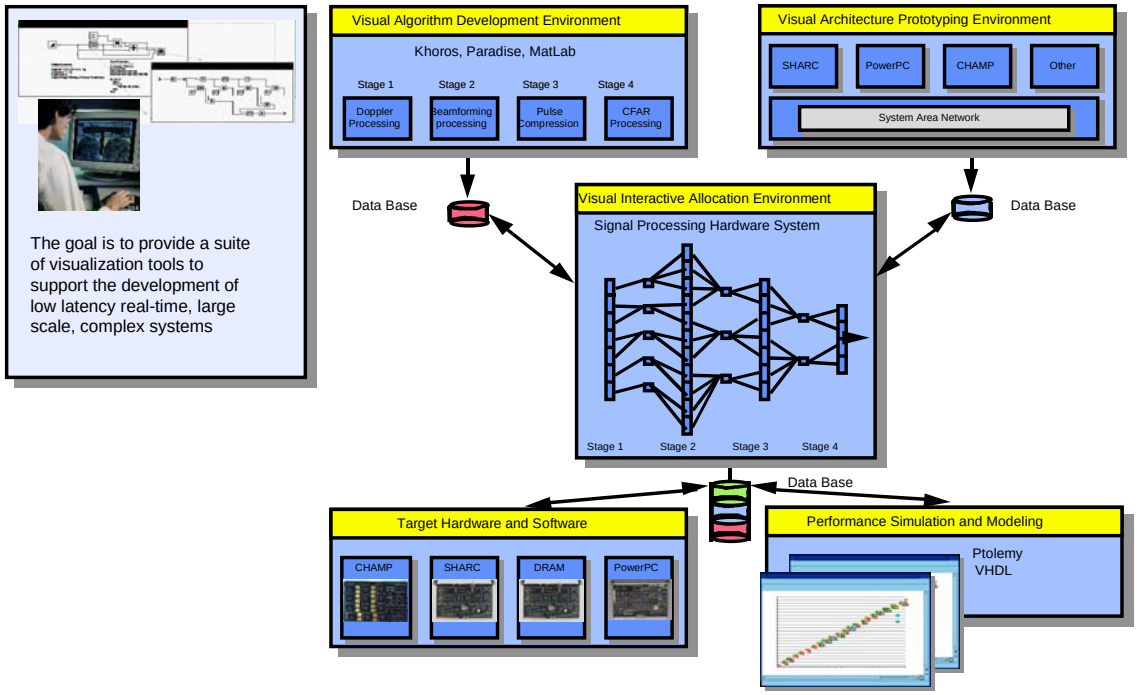



**Figure 4. EHPSCS Software Philosophy**

The selected application environments for the EHPSCS APU boards include workstations, commercial and ruggedized VME-based systems, and highly constrained real-time systems, such as missiles, radars, sonars, infrared search and track (IRST), and other dual-use systems. The common link between these diverse applications is the integrated modular hardware, software, interfaces, and design/debug tools.

Validation of the architectural concept was done with a demonstration of the functional testbed consisting of the processing-intensive Householder matrix transformation function that is part of the Space-Time Adaptive Processing (STAP) algorithm.

Upon validation, testbeds were produced and made available to the High Performance Computing community and its programs to support various system application developments.

### 3.3.1 Hardware Development Approach

The hardware development objectives under the EHPSCS program were to demonstrate the two-level multicomputer architecture concept implementation. Another objective was to develop an advanced packaging concept for embedded application transition of these technologies.

The approach taken for demonstration purposes was to leverage leading edge technology in the implementation of the two-level multicomputer. The Analog Devices SHARC 21060/62 processor was used as the core commodity processor for clusters of multiprocessing nodes which is linked with an embedded Myrinet switching network to enable scalable low latency, high bandwidth interprocessor communication. This concept is shown in Figure 5. EHPSCS Scalable Multicomputer. Two processing nodes, each consisting of a SHARC cluster and LANai Myrinet interface, were implemented on a single 6Ux160 VME eurocard form factor in the first realization of the two-level multicomputer, the Arithmetic Processing Unit (APU).

The complete EHPSCS testbed contains hardware in addition to the APU. A backplane was needed for network-level connectivity across APU modules. An adherence to COTS standards as much as possible drove the backplane with VME form-factor constraints for use in a COTS chassis. The backplane provides scalability within a chassis, but to expand on a system level and for communication with the workstation host a transition module was developed to support the Myrinet LAN protocol, for interchassis connectivity. The Myrinet Topology Expansion Module (MTEM) was developed to translate the SAN network-level connectivity across the backplane to the interchassis LAN protocol to scale from system to system. These components, in an industry standard, rack-mountable, 19-inch chassis, make up a complete hardware testbed.

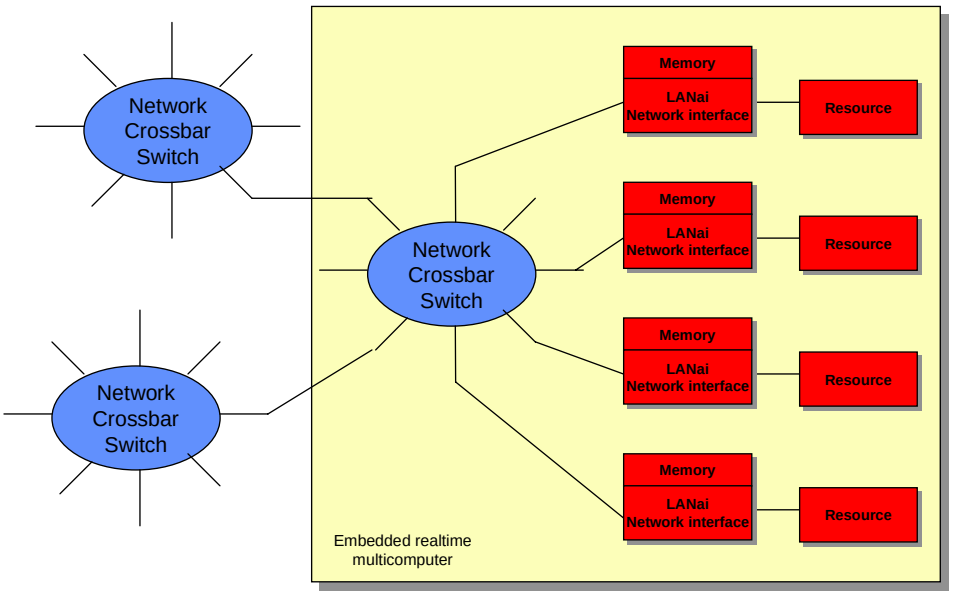



**Figure 5. EHPSCS Scalable Multicomputer**

The program investigated several advanced packaging concepts to not only demonstrate a version of the APU for highly constrained applications, but to address cost, power, and reuse issues. A low-cost High Density Interconnect (HDI) packaging technology, developed by Sanders, a Lockheed Martin Company and GE Corporate Research & Development allows a complete processing node, containing four SHARC processors, 3 MB RAM, and a Myrinet interface, to be implemented on a single 5 cm x 8 cm multi-chip module (MCM) package. For embedded applications, four MCMs can be implemented on a 6U x 160 VME eurocard form factor board, providing a total of 2 GFLOPS (or 43 MFLOPS/$in^3$) of peak performance on a single card slot. Leading edge HDI derivative technologies were used to experiment with subtile-based advanced packaging to investigate cost reduction, reuse of subtiles, and testability of non-known good die.

### 3.3.2 Software Development Approach

The software architecture approach for the EHPSCS program was to extend the full potential of the two-level multicomputer concept to enable system scalability, heterogeneous computing, fault tolerance, and innovative tools with industry standard support. To accomplish the scalable communication, high-performance, low-overhead software goals, an industry standard Application Programming Interface (API) was chosen for the program. The preferred API was Message Passing Interface (MPI).

The MPI specification for embedded applications was not finalized in time for the EHPSCS functional demonstration schedule. Therefore, Sanders continued on a parallel effort in the development of Data Synchronization Queues as an alternative to MPI for the demonstration and validation of the two-level multicomputer concept for embedded applications.

The other half of the parallel effort was the port of MPI to the EHPSCS testbed. Sanders Advanced Common Processor program worked cooperatively with the EHPSCS program in a research and development effort to port MPI and integrate MPI-based tools into the EHPSCS testbed. These efforts led to a series of realistic application demonstrations on the testbed for a more robust validation and application insertion opportunity for the EHPSCS technology.

A key attribute of this software architecture approach is to provide for a real-time network-level kernel that is free from node-specific operating system dependency. This is accomplished through a high performance, highly compact data synchronization kernel (referred to as DARC) that resides at the network layer of the two-level multicomputer architecture, shown in Figure 2.

DARC (Distributed Architecture Resource Controller) is a network runtime environment optimized for real-time parallel processing. In the context of the embedded EHPSCS program, DARC utilizes the Myrinet and its underlying control fabric of switches and flow control for point-to-point packet communication between network nodes. DARC executes on Myricom's LANai chip. The LANai forms the interface between Myrinet and each node. DARC has a front-end that interfaces directly to Myrinet for sending and receiving Myrinet packets and a back-end that interfaces directly to a node for resource control and data flow. The front-end validates incoming packets as DARC packets and hands them off to the back-end for processing.

Packets are generally of two types: resource control and data set. The portion of the back-end that handles resource control packets is very closely coupled to the hardware implementation of the node providing such functions as program loading and node debugging. The portion of back-end that handles the data set packets implements a unique parallel processing data-flow model called Data Synchronization Queues (DSQ).

The DSQ model represents an innovative data-flow approach to multiprocessor program development as opposed to the mainstream distributed control and shared memory approaches commonly employed by distributed real-time operating system (RTOS) kernels. With a distributed Real-time Operating System (RTOS), processors typically share memory and coordinate and synchronize their use of the shared memories through global semaphores. With DSQ, all data partitioning, data flow, and data synchronization information is encapsulated into a set of DSQ data structures that completely define how the processors will communicate at runtime. This approach lends itself well to a visual development environment for complex high-performance signal processing applications where a graphical user interface can automatically generate the DSQ data structures as they are interactively mapped by the user onto a network architecture.

Together, Myrinet, DARC, and DSQ form a network operating system for multiprocessor applications in the sense that together they manage all of the network nodes and resources, ensuring data flow direction, integrity, coherence, and timeliness without the need for processor-oriented operating systems. This allows the application programmer to concentrate on the computational aspects of a multiprocessor program without having to deal with the complexities of data flow and synchronization. Likewise, the nodes are spared the overhead of a distributed RTOS kernel, allowing more of each node's resources to be dedicated to the algorithm.

Figure 6 is an architectural decomposition of the prototype embedded EHPSCS testbed that utilizes the APU and SUN nodes. In this figure, each node is decomposed to illustrate the major hardware and software functional blocks and interfaces that comprise each node. The closely coupled shared memory interfaces within a node represent the first level in EHPSCS multi-computing and the loosely coupled Myrinet interface represents the second level in EHPSCS multi-computing. At the network interface is the LANai chip (it is referred to by Myricom as a chip and not a processor because it contains a processor and interface circuitry) that executes the DARC runtime and performs DSQ data flow. Layered between each node application program and the network is a Resource Network Interface (RNI) component. From the network point of view, a node contains a number of resources. For a resource to use the network, an RNI component must be created for that node.

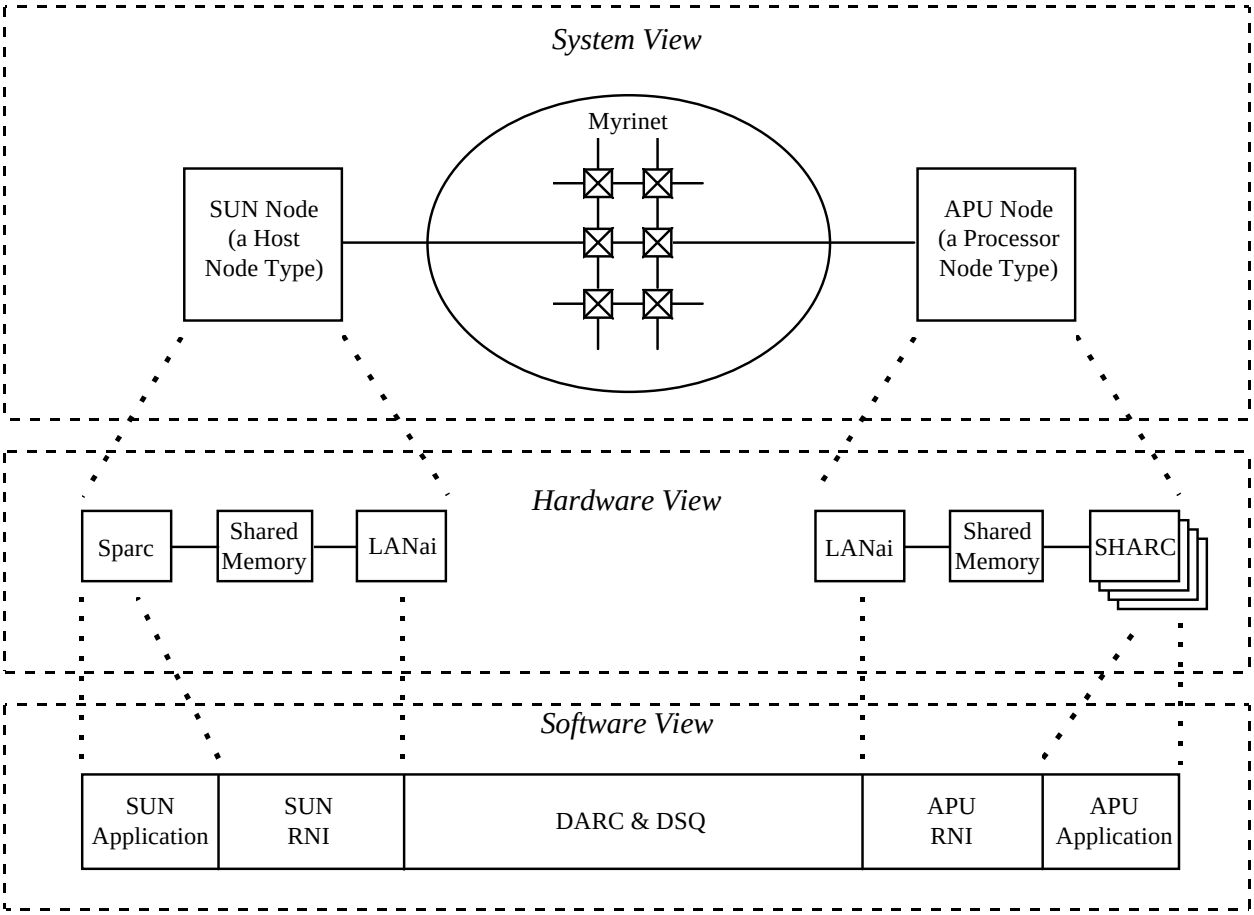



**Figure 6. Prototype Embedded EHPSCS Testbed**

The focus of this description is not on specific Myrinet nodes but on the enabling runtime environment that binds them into a network. Therefore, the focus is on the DARC control and DSQ data-flow software that spans the Myrinet from one LANai to another as illustrated in Figure 6.

A final note on terminology. The Myrinet connects nodes (the first level in multi-computing). Nodes in turn connect resources (the second level). Nodes can use their own resources (via level 2) or they can use resources on other nodes (via level 1). The term module encompasses the embedded aspect of the EHPSCS program. The distinction of a module is important because modules (circuit boards) plug into the embedded EHPSCS backplane that provides module-to-module Myrinet connectivity. One module may consist of multiple nodes interconnected on the module with Myrinet. For example, an APU module contains two APU nodes. Each APU node is further decomposed into a set of APU resources (LANai, SHARC1, SHARC2, etc.) as shown in Figure 6. Figure 7 illustrates all of these levels of EHPSCS embedding.

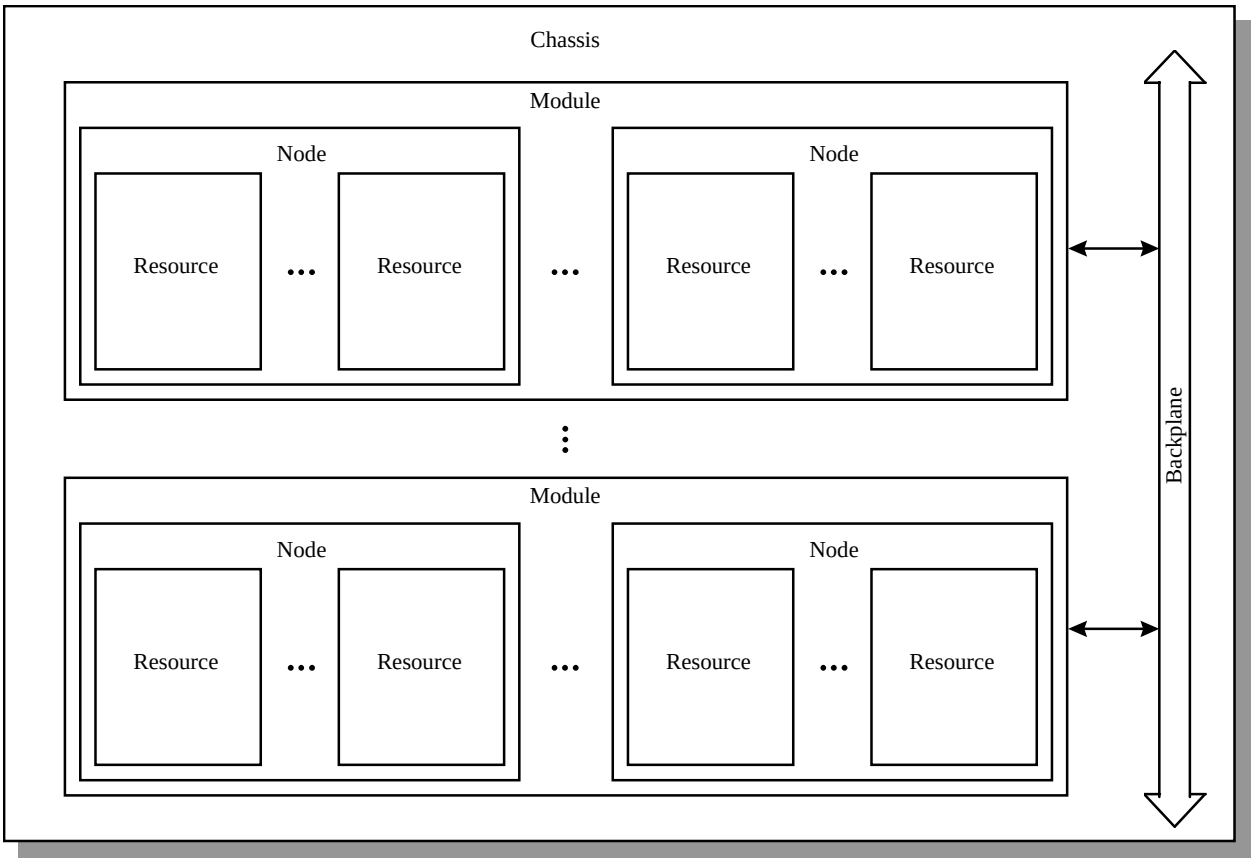



**Figure 7. Levels of EHPSCS Embedding**

In summary, the successful integration of Myrinet, DARC, DSQ, and the SUN and APU nodes is the primary software approach. A successful demonstration will result in the insertion of EHPSCS technology into COTS products in an effort to establish new standards in the area of embedded high performance computing.

### 3.4 Software Tools Approach

In keeping with the thrust of leveraging COTS technologies, the EHPSCS program integrated a set of commercially available system design, simulation, and profiling tools that would lead to a seamless path from algorithm implementation on workstations to real-time system operation on the EHPSCS testbed. Sanders has taken advantage of multiple ongoing DARPA-sponsored and academia developments to achieve an integrated environment to capitalize on successful High Performance Computing tools for application development, analysis and characterization, modeling and simulation, as well as software development.

Numerous COTS technologies and government-funded university research technologies have been integrated at Sanders to form the EHPSCS software development tool suite. The current EHPSCS APU reference implementation for the EHPSCS program utilizes the SHARC manufactured by Analog Devices, Inc. (ADI). ADI and other COTS vendors provide various SHARC-based development tools including compilers, multi-processor (MP) kernels, and emulators/debuggers.

Myricom provided board support software tools for interfacing host platforms with embedded Myrinet topologies and Sanders built on these tools to provide board support software for the EHPSCS APU.

Application programs targeted for execution on the multi-computer testbed were linked with libraries that provide the communication and DSP components required for runtime execution of massively parallel signal processing applications. For the communications library, the emerging Message-Passing Interface (MPI) standard (and implementations derived from that standard) were leveraged as a cost-effective solution. The MPI standard was developed under government-funded research programs at several cooperating universities and laboratories. MPI provides a common application program interface (API) for parallel programs ported between workstation clusters and embedded parallel machines. For the DSP library, a COTS implementation optimized for the SHARC and integrated onto the multi-computer testbed was used.

Rounding out the EHPSCS software development tool suite are a set of performance simulation and performance profiling tools. Again, existing public domain research programs were leveraged to provide these capabilities as a cost-effective solution.

Software from the Ptolemy project at the University of California at Berkeley was transferred and demonstrated as the EHPSCS multi-computer performance analysis tool. This software environment provides a performance simulation capability to model EHPSCS architectures on embedded Myrinet topologies. Two performance profiling tools based on MPI, Nupshot and ParaGraph were also transferred to the EHPSCS multi-computer testbed. Both Nupshot and ParaGraph are graphical program visualization tools for message-passing parallel computers. Nupshot was developed at Argonne National Laboratory and ParaGraph was developed at Oak Ridge National Laboratory.

### 3.5 ReConfigurable Transport Engine (RCTE) Approach

In the second half of the EHPSCS program, the baseline two-level multicomputer architecture was extended to specifically address multiple next-generation application challenges. Key challenges included technology-neutral support and hard real-time performance for wide range support of high-performance applications. Technology-neutral support enables continuous technology refresh and flexible system application through ready-insertion of the latest COTS technologies. Improvements in the hard real-time performance address applications with finer-grain performance requirements such as the STAP insertion program. The key objective of the RCTE design was to implement the extended architecture features in a prototype for characterization and concept validation.

The RCTE prototype design fully leveraged the development effort of the DARPA-sponsored Digital MicroArchitectures (DMA) program in 1996. Under the DMA program, Sanders designed the Common Logic Frame (CLF) network interface microarchitecture. This microarchitecture defined a set of approaches that were adapted for the RCTE prototype. These approaches included a hardware-based network protocol acceleration engine, zero-copy message passing, and reconfigurable network and resource interfaces to support different COTS technologies.

The RCTE prototype is based on a low-cost commodity RISC engine combined with field-programmable gate-array (FPGA) technology and a Myrinet network protocol interface to demonstrate reconfigurability in network interface and technology independent supports. The RCTE prototype interfaces to the existing EHPSCS testbed environment to facilitate test and integration and characterized for hard-real-time performance application. The approach for the RCTE under the EHPSCS program was to functionally test and validate the hardware prototype and demonstrate and characterize performance that can be compared to similar measurements on the LANai-based APU.

# 4 Progress

The EHPSCS hardware/software functional testbed system along with a released software development tools environment and documentation provided insertion program users with a complete operating environment for advanced application developments requiring scalable high performance computing technology. To date, Sanders has delivered 14 EHPSCS testbed systems which include 70 APUs and 18 MTEMs, to multiple insertion programs including UYS-2A, STAP, ACP, UUV, P507, and SHARE. Significant progress has been made by the insertion program users to date in the porting and mapping of advanced algorithms on the EHPSCS platform. Examples of insertion accomplishments include the successful demonstration of a 64-processor EHPSCS system for space-time adaptive processing in May, 1997 by the STAP program as well as the detection and tracking algorithms running in the SAR, EO, and IR processing domains of the ACP project.

The EHPSCS testbed system developed under the EHPSCS program is an expandable configuration providing up to 11 GFLOPS peak performance using 1 to 11 EHPSCS APU modules. As Figure 8 shows, the EHPSCS functional testbed consists of the following components:

- Standard 20-slot VME chassis with 13-slot EHPSCS backplane section and 6-slot standard VME backplane section for support of COTS product options.
- Arithmetic Processing Unit (APU) with low-cost 6U-160 implementation using discrete components; SHARC-based, dual-node capability providing 960 MFLOPS peak performance; on-board 128 Mbyte memory expansion; on-board Myrinet 8-port network switch; four Myrinet ports on the backplane connector; two Myrinet SAN ports on the faceplate; on-board boot support, and JTAG support.
- Myrinet Topology Expansion Module (MTEM – refer to Section 4.1.3), which provides Myrinet topology expansion within the EHPSCS backplane configuration and external to the backplane for additional EHPSCS chassis and/or workstations. The MTEM also provides redundant network configuration support within the EHPSCS testbed. Four Myrinet LAN protocol ports are provided by each MTEM board.
- System software and application development environment with Sun workstation host support over a common Myrinet network, a high-performance network kernel based on the Data Synchronization Queue (DSQ) technique, Myrinet system boot-up support, system application analysis and development tools, SPOX-MP COTS kernel support, and SHARC target development tools.

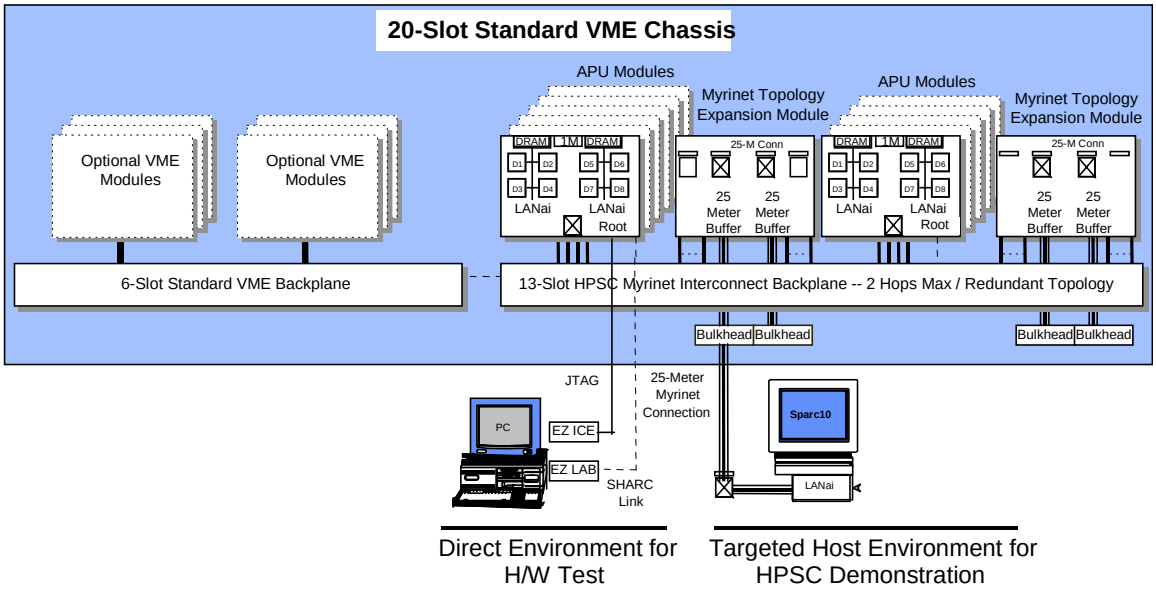



**Figure 8. EHPSCS Testbed Functional Block Diagram**

The functional demonstration was completed in June 1996. The demo consisted of the processing-intensive Householder matrix transformation function that is part of the Space-Time Adaptive Processing (STAP) algorithm. The Householder function was allocated to a parallel SHARC processor node on an APU module. Real-time sensor data sourcing is simulated by a second node with data routed through the Myrinet network. The functional demonstration validated the EHPSCS two-level architectural concept, the functional testbed hardware and software designs, and the software development tools.

In October 1996, the eMPI version for resource-constrained systems was also demonstrated on the EHPSCS testbed under the coordinated research and development effort with the ACP program using an industry standard API. The EHPSCS MPI implementation offers EHPSCS users direct portability of advanced applications developed under workstation or supercomputer environments to the EHPSCS parallel target environment.

In November 1995, Sanders demonstrated the first advanced packaging development with the 5V Embedded DSP Multichip Module, which provided a path for insertion of the EHPSCS technologies into highly constrained applications. That was followed in November 1996 by the completed design of the 3-volt version of the 480-MFLOPS node design targeted for MCM packaging. The 3-volt design is essential in reducing system power consumption for large computation system applications and can be transitioned to system application with aggressive processing density requirements. The 3-volt design was also the basis of an MCM exercise to evaluate tile-based packaging concept completed in July 1997.

In July 1997, an alpha version of a Multi-Processor Source Debugger based on Dolphin Interconnect Solutions' TotalView commercial tool was demonstrated on the EHPSCS testbed. TotalView is a leading COTS MP source-level debugger product. The integrated TotalView capability offers EHPSCS users a highly productive network-based multi-processor debugger

environment. Also integrated into the EHPSCS tool suite is the commercial White Mountain DSP SHARC Emulator tool which provides EHPSCS users with a high degree of flexibility in choosing a preferred debug environment.

In April 1998, the RCTE network interface controller prototype was completed and characterized to validate the microarchitecture concept and to demonstrate reconfigurability and hard real-time performance improvements.

The sections that follow describe in detail the progress of each development effort for the EHPSCS program.

### 4.1 Multicomputer Testbed Development

A first-generation EHPSCS hardware/software functional testbed using discrete components was completed in June 1996. The EHPSCS functional testbed provides up to 11 GFLOPS of peak performance in a single COTS VME chassis using eleven APU modules.

The testbed consists of a standard 20-slot 6Ux160 VME chassis with a 13-slot EHPSCS backplane section and a 6-slot standard VME backplane section for support of COTS product options. This backplane and chassis are available from a variety of commercial vendors. The EHPSCS backplane is a 13-slot backplane providing mainly Myrinet and SHARClink connectivity for up to 11 APU cards and 2 MTEM cards. The combination of the two backplanes offers maximum network throughput, an open architecture, and COTS module support. The testbed is shown in Figure 8 and Figure 9.

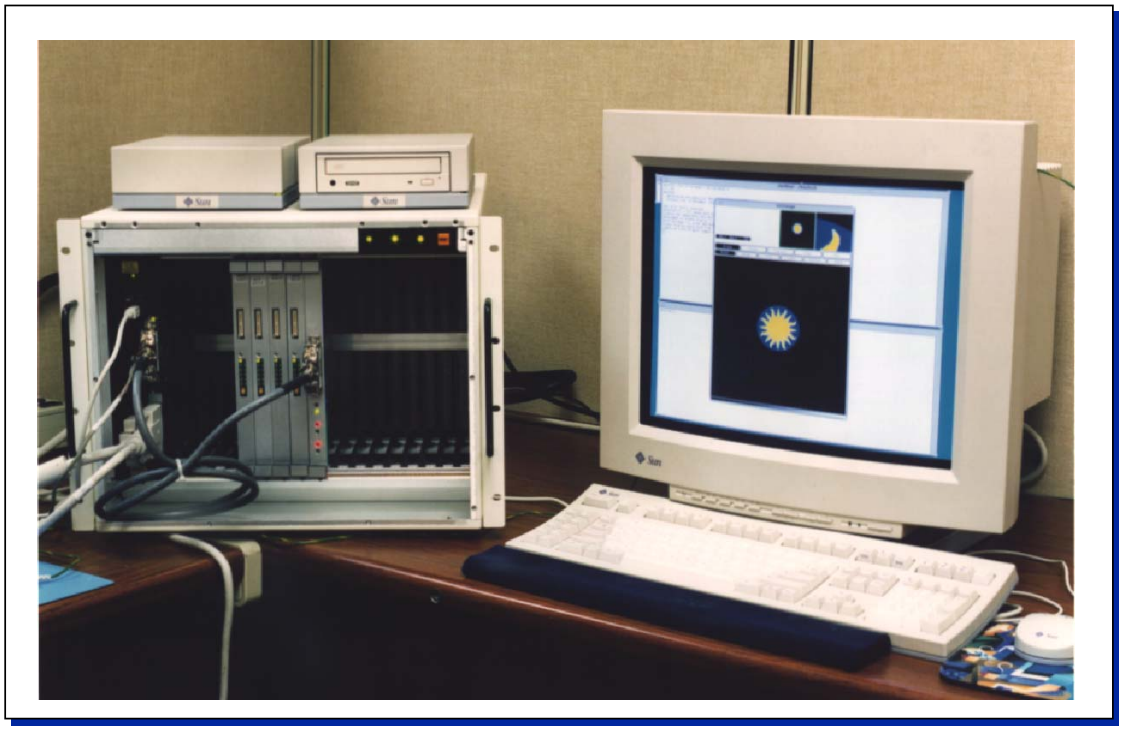


**Figure 9. EHPSCS Testbed Photo**

The typical development configuration for the testbed is a quantity of APU boards combined with at least one MTEM card and a host computer. The MTEM card provides a LAN connection to the host computer equipped with a Myrinet interface card. The host can be a workstation or a VME single-board computer and provides a host environment for application development. A PC is connected to the testbed via SHARC Links for hardware test and verification of the SHARC-based APU boards.

### 4.1.1 Arithmetic Processing Unit Development

The first implementation of the EHPSCS two-level multicomputer is the Arithmetic Processing Unit (APU). The APU module utilizes the Analog Devices

ADSP-21060/21062 single-chip Digital Signal Processor (DSP) as the application processor. This Super Harvard ARchitecture Computer (SHARC) is a 32-bit processor optimized for signal processing applications. The LANai from Myricom is used for the network controller function on the APU which implements the Myrinet switched network. As shown in Figure 10, the design consists of two identical processing nodes of four ADSP-2106x DSPs and the Myrinet LANai providing 960 MFLOPS peak processing throughput. The two processing nodes are connected through the Myrinet via an on-board 8-port crossbar switch.

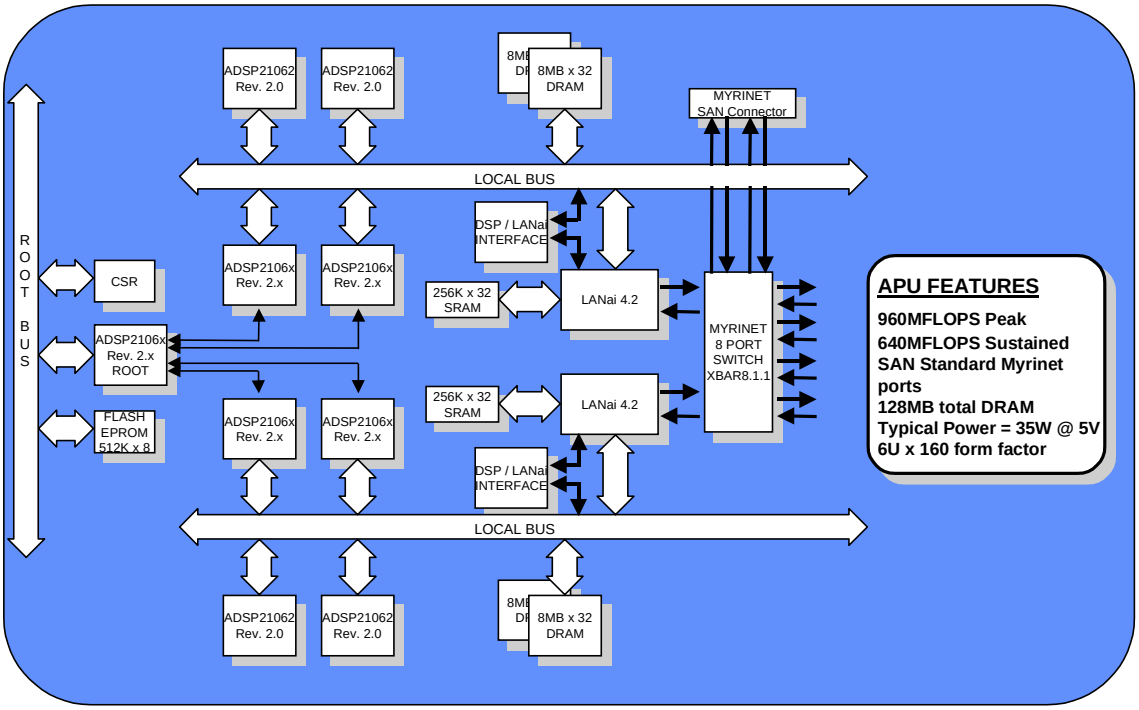



**Figure 10. EHPSCS Arithmetic Processing Unit Block Diagram**

The EHPSCS APU module utilizes a 6Ux160 VME form factor platform for printed circuit board structure. The board is shown in Figure 11. Each processing node contains up to 64 MB of DRAM and 1 MB external SRAM. The board has link port connectivity and JTAG access for testability as well as COTS software tool support. The SHARClinks and Myrinet ports run out

of high-density, 235-pin connectors, which provide a high degree of I/O capability to the backplane for flexible, high-performance applications. For more detail on the functionality and operation of the APU board, please refer to the attached Hardware Description Document for High Performance Scalable Computing Arithmetic Processing Unit Revision 1.

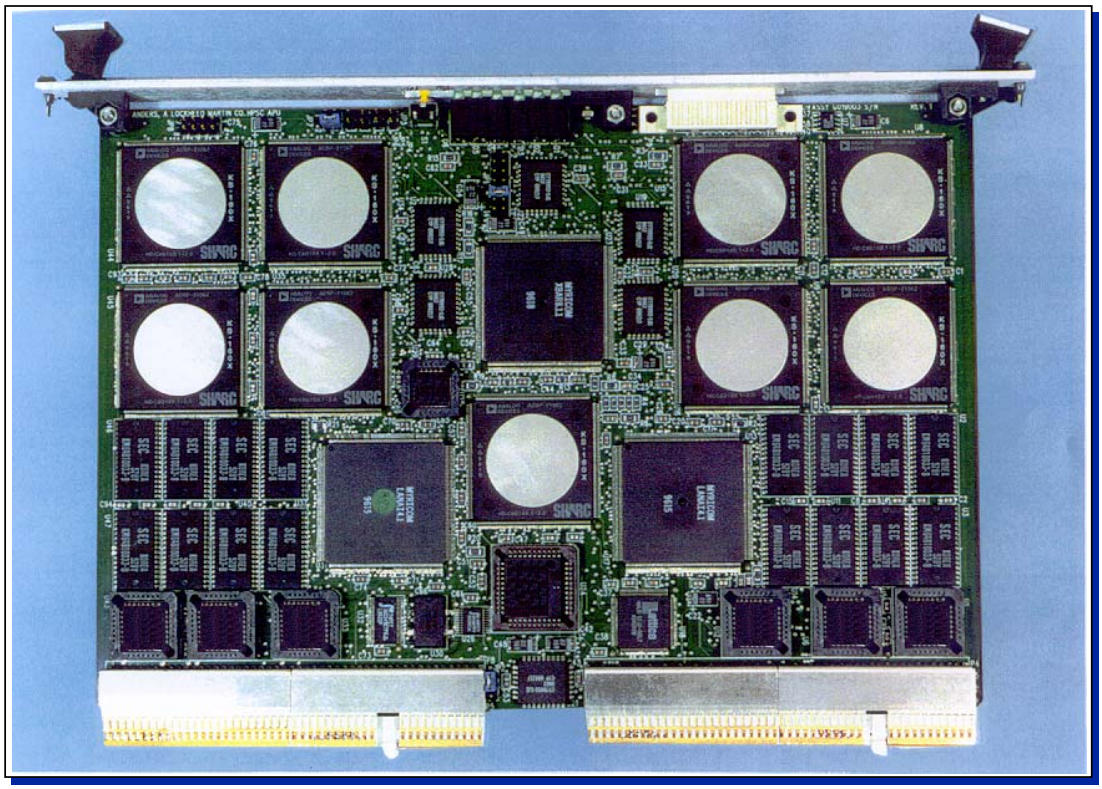


**Figure 11. EHPSCS Arithmetic Processing Unit**

The APU module was successful in demonstrating and validating the two-level multicomputer architecture and providing a software development testbed. A suite of SHARClink-based confidence tests were developed to verify the hardware in an automated fashion using the Arithmetic Processing Unit Test Procedure, which is attached to this document. These tests allowed some in depth exercise of the APU hardware. This exercise revealed a subtle fault at 40 MHz operation that could be corrected with a performance increase in the LANai's local bus. Myricom cooperated with Sanders in determining the source of the failure and enhanced the local bus drivers in a respin of the LANai die.

### 4.1.2 EHPSC Backplane Development

The SHARC-based portion of the APU design was derived from a legacy SHARC-based VME board design developed under Lockheed Martin IRAD. In an effort to leverage COTS technology and reduce design risk, the original concept of an APU implementation included the VME64 bus interfaced to the root processor. A custom P2 backplane was investigated to provide Myrinet connectivity as well as link port and JTAG connectivity. The SHARC links were used extensively in the legacy design for test and interprocessor communication. These were included

in the APU design to reduce test and integration time while working with the new Myrinet. It soon became evident that the VME standard on P2 could not support the quantity nor signal integrity necessary for these I/O requirements. With the addition of high-density, impedance-controlled backplane connectors to the APU, the VME bus was no longer necessary and was removed from the design concept. Thus, a custom backplane mainly consisting of redundant Myrinet connectivity and link port connectivity was required and developed. The backplane provides the high bandwidth and signal integrity as well as a bridge to the COTS VME backplane resident in the same chassis. It retains the VME standard board-to-board pitch and compatibility with standard chassis.

The major backplane connectivity is shown in Figure 12 and Figure 13. Each arrow in Figure 13 represents a bidirectional Myrinet port between an APU module and an MTEM module. Similarly, each arrow in Figure 12 represents a bidirectional link port between DSPs. Note that Figure 12 shows only board to board link connectivity. The on-board connectivity is not represented.

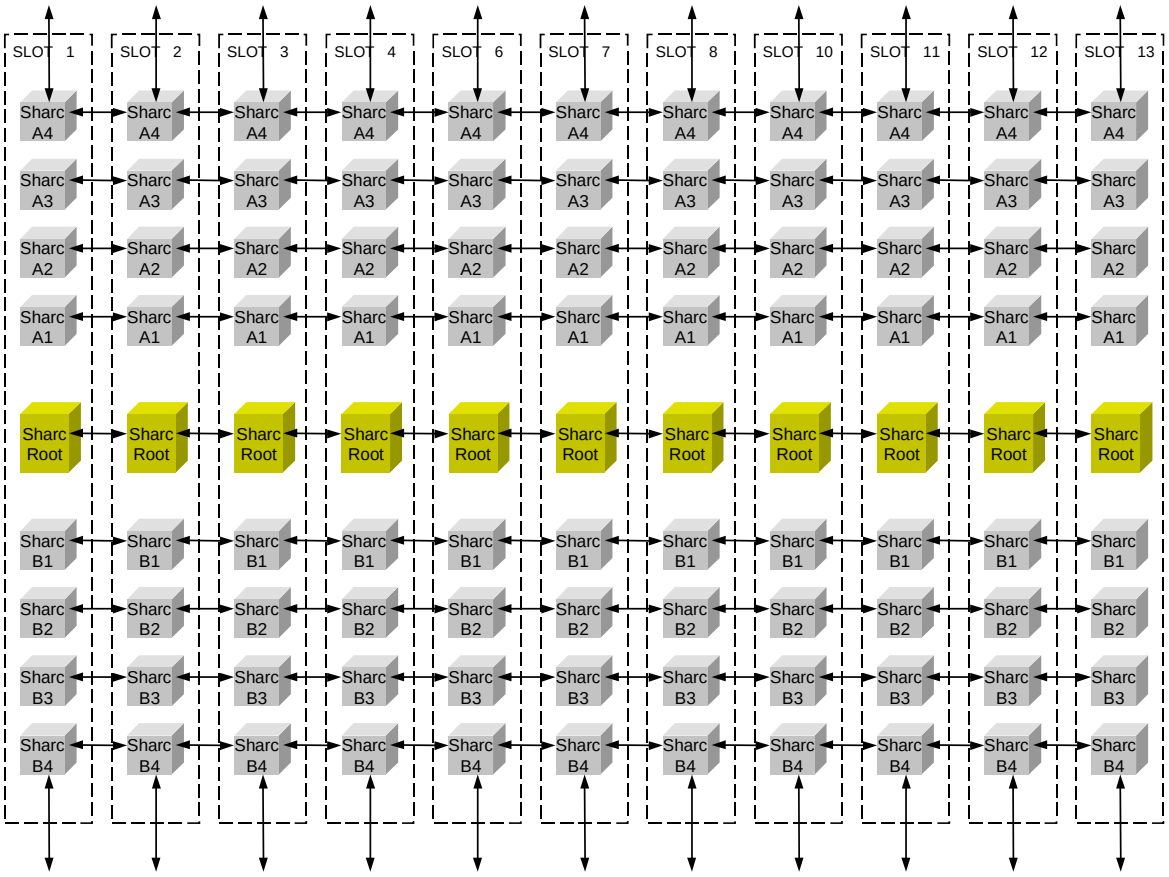



**Figure 12. EHPSCS Testbed Backplane SHARClink Connectivity**

Recent progress has been made in the industry standardization effort of Myrinet System Area Network (SAN) connections. Midway through the EHPSCS program, Myricom addressed the standardization approach in two ways. The first was a Myrinet standard connector/cable definition that supports two bidirectional SAN ports. These offer board-to-board cable connectivity within a chassis, most commonly via front panel connectors. The second was an

industry draft standard for Myrinet SAN links on VME P0. This draft standard defines Myrinet connectivity supporting two bidirectional SAN ports over the VME P0 connector. This draft is Myrinet-on-VME Protocol Specification Draft Standard, VITA 26-199x, Draft 0.5 and can be viewed at http://www.vita.com/vso/draftstd/myri-vme-d05.pdf, a copy of which is attached to this document. The front panel cable/connector was implemented on the APU, Rev. 1.1.

In support of the P0 draft standard, CSPI Inc. has since offered standard products for backplane overlay modules that plug into the P0 connector across four slots of a VME backplane to provide Myrinet switching among all eight ports. Refer to http://www.cspi.com/multicomputer/2600tech.htm for a detailed description of the use of this standard and the backplane overlay module.

### 4.1.3 Myrinet Topology Expansion Module Development

The Myrinet Topology Expansion Module (MTEM) was developed in cooperation with the ACP program as a means to increase communication bandwidth among APU cards, to provide network redundancy within the testbed, and to provide network protocol conversion to a LAN to extend the system interconnect beyond a single cluster of APU modules.

The MTEM board consists of two 8-port crossbar switches that switch 13 Myrinet SAN ports, one from each APU and two from the other MTEM via the backplane, and three MI protocol converters. The testbed supports two MTEM modules. These modules provide the network redundancy as each provides switching to all 11 APUs in a testbed. Figure 13 is a comprehensive diagram showing all testbed Myrinet connectivity between the 11 APU boards and the two MTEM boards. It includes the connectivity among the backplane, within the MTEMs, and within the APUs. All arrows in the figure represent Myrinet SAN ports.

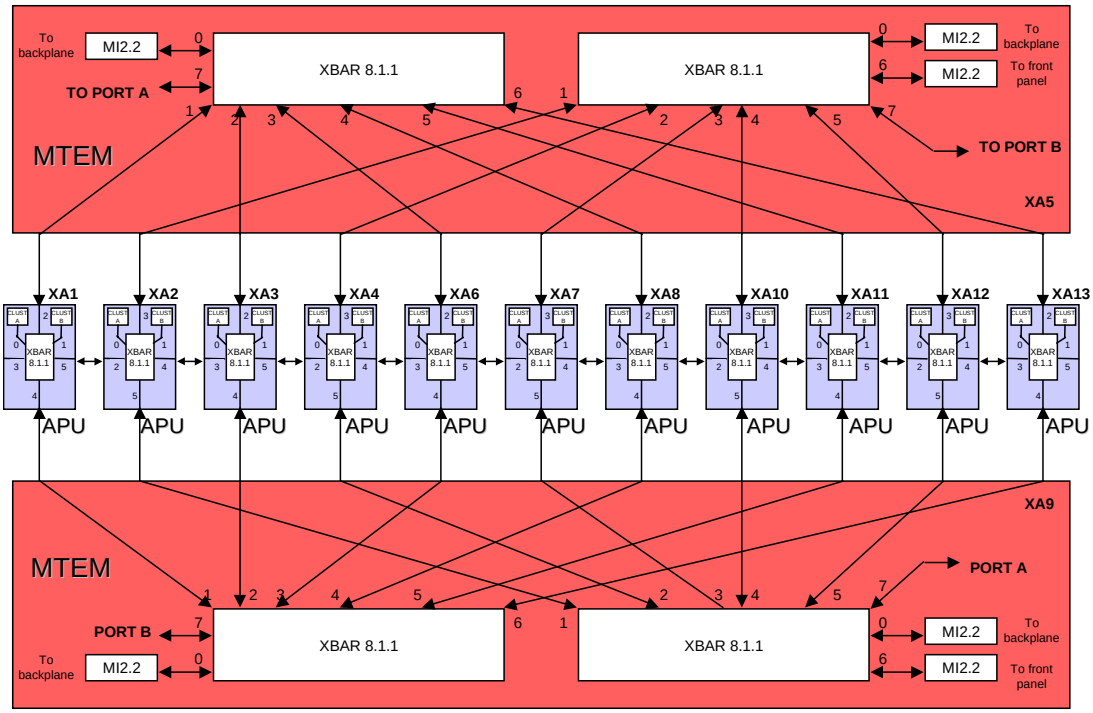



**Figure 13. EHPSCS Testbed Myrinet Connectivity**

The MTEM boards provide a peak cross-sectional LAN bandwidth into and out of the testbed of 1.92 Gbytes/sec, which facilitates the testbed-to-testbed scalability of the EHPSCS architecture. The MTEM board is shown in Figure 14.

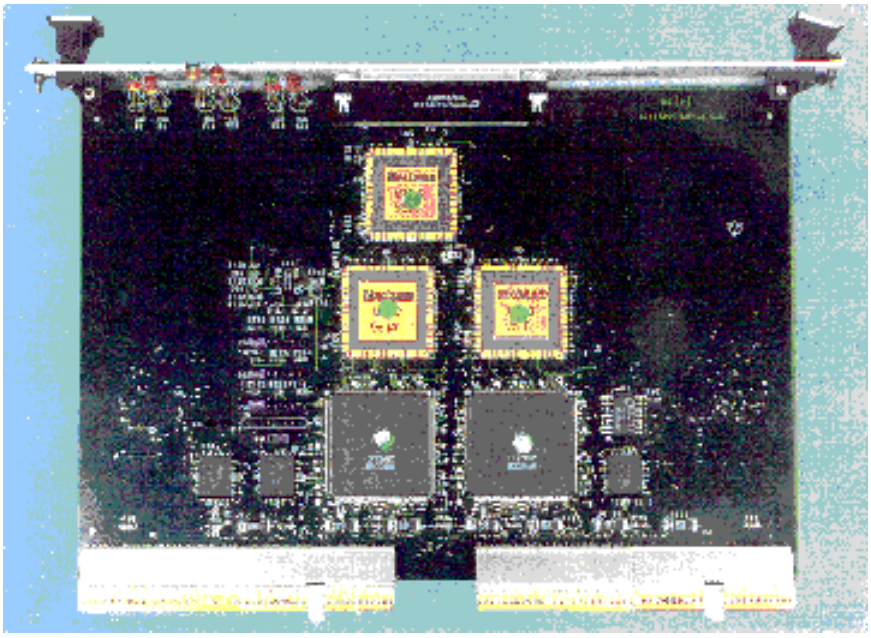


**Figure 14. EHPSCS Myrinet Topology Expansion Module**

### 4.1.4 Distributed Architecture Resource Controller Development

The Distributed Architecture Resource Controller (DARC) is a Myrinet control program that implements the network protocols established for the embedded EHPSCS system. The DARC is cross-compiled from C source code on a host Unix development platform and loaded onto an embedded network of LANai processors via a host Myrinet interface card. The interface card contains a LANai that is loaded with a DARC image using tools on the host. The LANai on the host interface card is called the root. The embedded network LANais are booted from ROM when the network nodes are powered up. The network LANais execute a boot loader that receives and executes a DARC program image from the network. The DARC program images are distributed onto the network nodes via the host node's root LANai.

Every unique type of node in the embedded EHPSCS system will receive and execute a unique DARC program image. For example, the SUN host node executes the SUN-DARC program and the APU processor node executes the APU-DARC program. The back-end portion of the DARC that interfaces to the resources on a node is what makes each DARC unique to a node type. The front-end portion of the DARC that interfaces to the Myrinet implements a common EHPSCS packet communication protocol. A top-level block diagram showing the software components of the DARC software is illustrated in Figure 15.

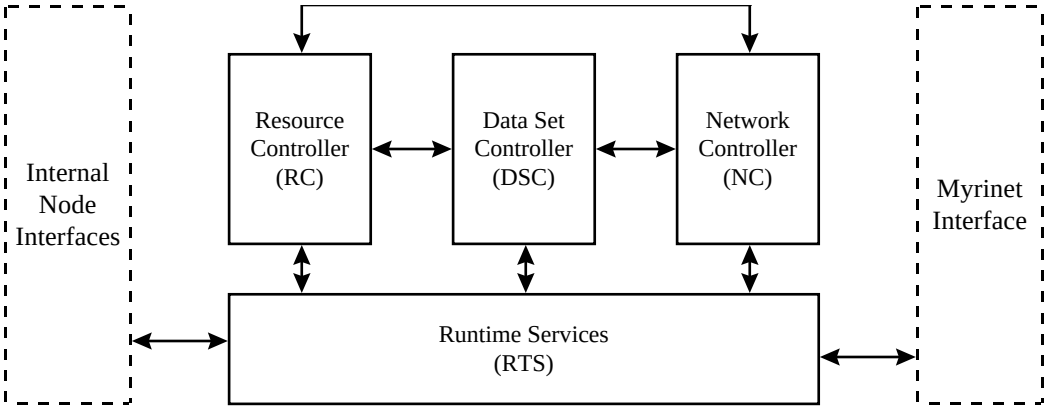



**Figure 15. DARC Software Top-Level Block Diagram**

The DARC functionally decomposes into four primary software components: the Network Controller (NC), the Resource Controller (RC), the Data Set Controller (DSC), and the Runtime Services (RTS). The RTS is the underlying software that handles boot loading, context switching, memory management, and hardware interface drivers. The NC, RC, and DSC are users of the RTS.

The NC is responsible for both input packet processing and output packet processing. The NC uses the RTS to send and receive packets on the Myrinet. The NC decodes incoming packets as either resource control messages or data set messages and passes the messages to the RC and DSC, respectively, where the messages are decoded and processed. Output messages constructed by the RC and DSC are passed to the NC where they are constructed into Myrinet packets and transmitted.

The RC is responsible for handling resource control messages. The RC uses the RTS to interface to the internal node interfaces and obeys the hardware and software protocols established for those interfaces. Resource control services include boot loading, program loading, data loading, data extraction, resource debugging, etc.

The DSC is responsible for handling data set messages. The DSC also uses the RTS to interface to the internal node interfaces and obeys the hardware and software protocols established for those interfaces. The DSC and RC internally interface with each other to arbitrate access to the shared resources. Data set services include input and output data set construction, queuing, and notification to local and remote resources. Data sets and Data Synchronization Queue (DSQ) data flow are described in detail in Section 4.1.5.

In order to fully understand the structure and execution environment of a DARC program on an EHPSCS node, it is helpful to describe the DARC by a specific example. Currently, two DARC implementations exist. The SUN-DARC, which executes on a host node, and the APU-DARC, which executes on a processor node. The interface to the SUN-DARC from an application program (resource) executing on a SUN is documented in the SUN Resource Network Interface Component Specification, which is attached to this document. The interface to the APU-DARC from an application program (resource) executing on an APU is documented in the APU Resource Network Interface Component Specification, also attached to this document.

### 4.1.5 Data Synchronization Queue Development

Data Synchronization Queues (DSQ) is a data-flow synchronization model for real-time parallel processing. It is implemented within the DARC runtime software, forming the back-end to an embedded EHPSCS Myrinet node. In an embedded EHPSCS system, the DARC runtime handles the data flow and DSQ maintains the data structure. The application program distributed across the compute resources within a node performs the data processing and works on a group of network data sets.

From the network viewpoint, each compute resource on a processor node has a unique set of input data sets and output data sets. Data may be scattered from one resource to many, or data may be gathered from many resources to one. Data scattered from one resource to many resources is first constructed into a complete data set before transmission. Likewise, data gathered from several resources into one resource is constructed into a complete data set before notification to the application executing on the resource. These scatter/gather operations provide data synchronization between the compute resources.

Additionally, a data set is not notified to the application until the application requests the data, therefore data sets may be queued. This data queuing provides a buffer between the network and the application. A different queue will exist for each unique type of data set. The length of a queue will depend on the memory available on a resource and the worst-case expected latency that an application may experience before requesting the data. Queue length may be tuned on a per resource basis and is dependent on the application program.

In DSQ, an application executing on a compute resource is simply a compute engine that receives data, processes the data, and transmits the results. The compute engine is not concerned how the data gets there or where it is going to. The compute engine is concerned only that data structure and timeliness are maintained so that it may process the data in a coherent and timely manner. All of the parameters that describe the input and output data sets for each resource in a DSQ application are specified in a pair of input and output parameter tables, one pair per node. The parameter tables encapsulate all data structure, data timeliness, and network data flow information.

The parameter tables are constructed by the application programmer and executed by the DARC at runtime. Parameter tables are created either manually or by an automated visualization tool. Other tools already exist to load and execute application programs and parameter tables. For each resource integrated into the EHPSCS system, an RNI specification will exist that specifies how the resource application program will access the data sets at runtime.

#### 4.1.5.1 Data Sets

A data set represents a single coherent block of data that exists on a single node. It is coherent both logically and physically. A data set is logically coherent in that it contains a logical grouping of data for computation by the algorithm resident on the resource. A data set is physically coherent in that it is bound to a physical address in a single contiguous block of memory. A data set exists on a node as either an input to or output from the application program executing on the resources. Figure 16. Data Set Organization is an illustration of the different

ways that a data set may be perceived. In all three cases the data set resides at physical memory location M and is B bytes in length.

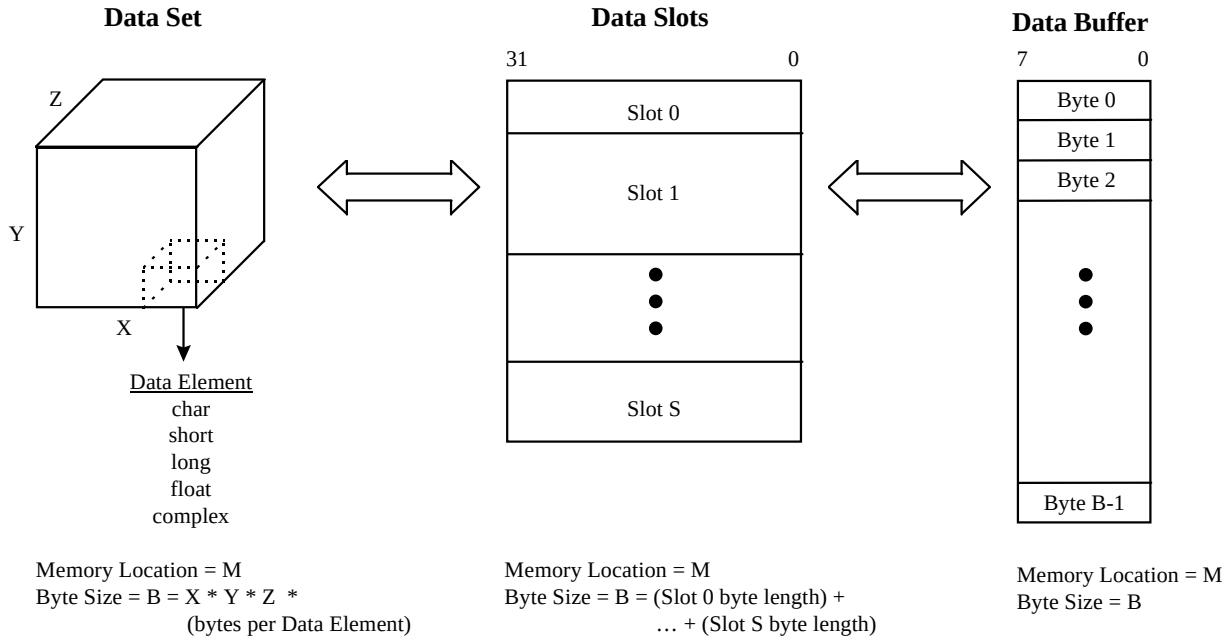



**Figure 16. Data Set Organization**

From an algorithmic viewpoint, a data set comprises an X, Y, and Z component. A single vector would then be described as {X, 1, 1}, a matrix as {X, Y, 1} and a three-dimensional (3-d) array as {X, Y, Z}. A single element of a vector, matrix, or 3-d array is described by its element size, E. The basic minimal addressable unit is an 8-bit byte. A fully constructed data set is presented to an application on a resource (input data set) or to a resource on the network (output data set) when all of its data slots have been filled. Data slots will not be filled until the data buffer has been allocated for use by a data set.

From a network viewpoint, a data set comprises S data slots of variable length with a fixed 32-bit word width. A data set is partitioned into slots for the purposes of scatter/gather operations. Each data slot represents a portion of the data set that will be transmitted or received (depending on whether it is an input or output data set) to or from other network resources. Data slots are inserted into Myrinet packets that travel the network from resource to resource. A packet contains the slot data plus packet header information for traversing the Myrinet. The definition of a data slot actually binds the application's data to a physical location in resource memory. For an output data slot, the definition also binds the slot to (1) a trigger condition indicating when the slot will be transmitted, (2) a physical path through the network, and (3) the data set and data slot on the receiving resource.

From a hardware viewpoint, a data set is simply a buffer of B bytes of data located at memory location M. Data buffers are allocated statically at system startup when the parameter tables are loaded onto each resource. The parameter tables specify only offsets into the data buffers so that individual slots may be located precisely at runtime, given the static base addresses computed at

startup. Typically, the data buffers will be located off the LANai's E-Bus in a memory, such as a large DRAM, that is shared with the node's resources. The parameter tables that point into the DRAM are typically located off the LANai's L-Bus in an SRAM that contains the DARC runtime software.

The DARC runtime software performs two fundamental operations with data sets: data set instantiation and data set notification. Instantiation is the process of loading parameter tables, allocating buffers, forming queues, etc. prior to the application's execution. Notification occurs during the application's execution. Notification is the process of filling slots, setting timers, and presenting complete data sets to the local application or to another resource on the network. Notification also involves the detection and notification of timeout conditions or misrouted data slot packets to a designated system controller resource.

The application program may require several different types of data sets to perform its algorithm. The various parameters that describe a data set type and how it is instantiated and notified are highlighted below.

*Data Set **Class*** — A data set is either an input to a resource or an output from a resource. All data sets for the entire system are classified as belonging to either the class of input data sets or the class of output data sets.

*Data Set **ID*** — Every data set within a class receives a unique ID that is simply an enumeration starting at 0. The class of input data sets begins at ID=0 and the class of output data sets begins at ID=0.

*Data Set **Queue Length*** — This parameter specifies how many data sets buffers will be allocated for queuing data sets of this type (class and ID). If this count is exceeded, then a notification will be sent to a designated system controller resource indicating a data set slot was received but dropped. This parameter may be tuned to fit the resource's memory and the application's timing.

*Data Set **Dimensions*** — The logical X, Y, Z, and E dimensions of a data set must be specified to accurately size and locate the data set buffers during data set instantiation. These parameters are passed to the application program during data set notification.

*Data Set **Partitions*** — A data set is partitioned into a number of variable-sized slots. This parameter specifies how many slots the data set is partitioned into. A data set slot (DSS) is identified sequentially starting at a DSSID=0. Each individual slot will carry its own DSS parameters as defined below.

*Data Set **Notifications*** — For input data sets, this parameter indicates which compute resources located on a node (a processor node may contain several compute resources) are to receive a completed data set. For output data sets, this parameter indicates which compute resources must contribute to an output data set before the data set is transmitted.

*Data Set **Timeout*** For input data sets, this parameter is used to determine if a data set has arrived within a given timeout interval. A watchdog clock is started when the first data slot is received and checked on the arrival of other data slots. If the data slots have not completely filled the data set before the timeout interval, then a notification will be sent to a designated system controller resource indicating a data set timeout has occurred.

The data set slots are further described by the following parameters:

*Data Set Slot **ID*** Every data set slot within a data set receives a unique ID that is simply an enumeration starting at 0.

*Data Set Slot **Size*** This parameter is the number of 32-bit words of data in this slot.

*Data Set Slot **Offset*** This parameter is the memory offset into the data buffer associated with the data slot and data set.

*Data Set Slot **Routing*** For output data set slots, the routing parameters specify the number of resources that will receive this data set slot as well as the Myrinet route codes required to reach those resources from this resource.

*Data Set Slot **Notification*** For output data set slots, the notification parameters specify the receiving resource's input data set ID and input data set slot ID that will receive this output data set slot.

#### 4.1.5.2 Queuing and Synchronization

The discussion to this point has focused on the organization and parameterization of data sets which are the fundamental DSQ data structure. This section will form data sets into data queues and provide a data-flow synchronization structure for the queue elements.

Figure 17 shows the composition of a single DSQ. One DSQ will exist for each unique data set defined in the network application. A single data set is buffered in a data buffer. A group of data buffers forms a data queue (DQ). A data synchronization table (DST) is associated with each DQ. All of the parameters that define a data set, data buffer, and data queue are contained in the DST. The following mathematical notation represents a single data synchronization queue: DQ + DST = DSQ.

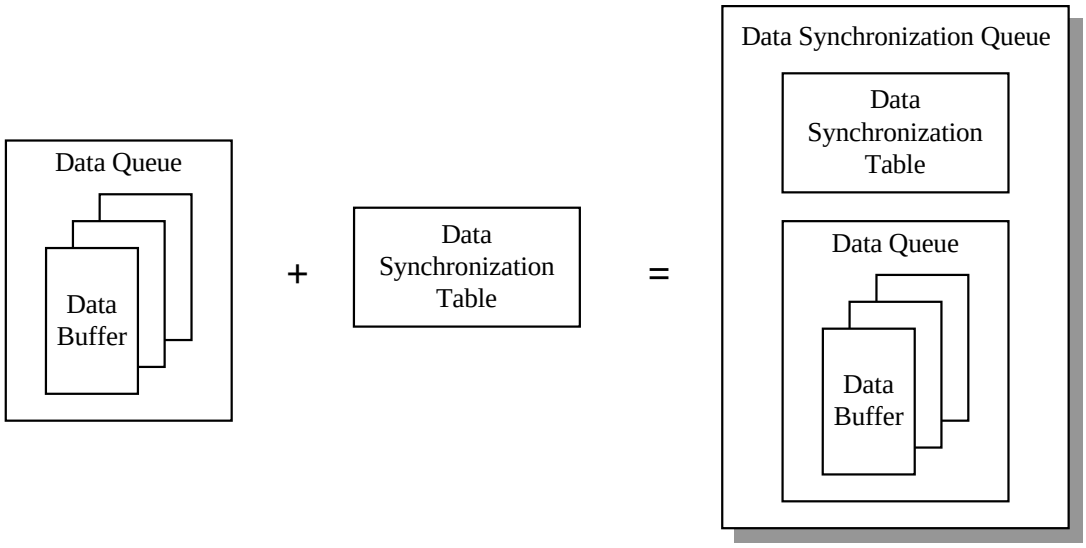



**Figure 17. One Data Synchronization Queue**

Each DQ, DST, and DSQ may be further classified as belonging to an input (I) class or output (O) class, resulting in the following notation:

| | | | |
|---|---|---|---|
| Input queuing and synchronization: | IDQi | + IDSTi | = IDSQi |
| Output queuing and synchronization: | ODQo | + ODSTo | = ODSQo. |

In this notation, i ranges from i=0 to i=N, where N is the total number of input data sets in the network application and o ranges from o=0 to o=M, where M is the total number of output data sets in the network application. Remember, a unique data set within the network is classified first as either an input or output data set and then it is given a unique enumeration number within its class. Any given node in the network will own a unique group of IDSQ's and ODSQ's for the resources it services.

Figure 18 brings together all of the data structures and terminology defined so far to illustrate the DSQ runtime environment maintained by the DARC software executing on a node's LANai chip. At the Myrinet interface are the input packets that are received by DARC and the output packets that are sent by DARC. As described earlier, packets contain either control messages or data messages. The DSTs comprising the parameter tables enter as control messages within input packets. The data slots comprising the data sets in the data queues enter and exit as data messages within input and output packets.

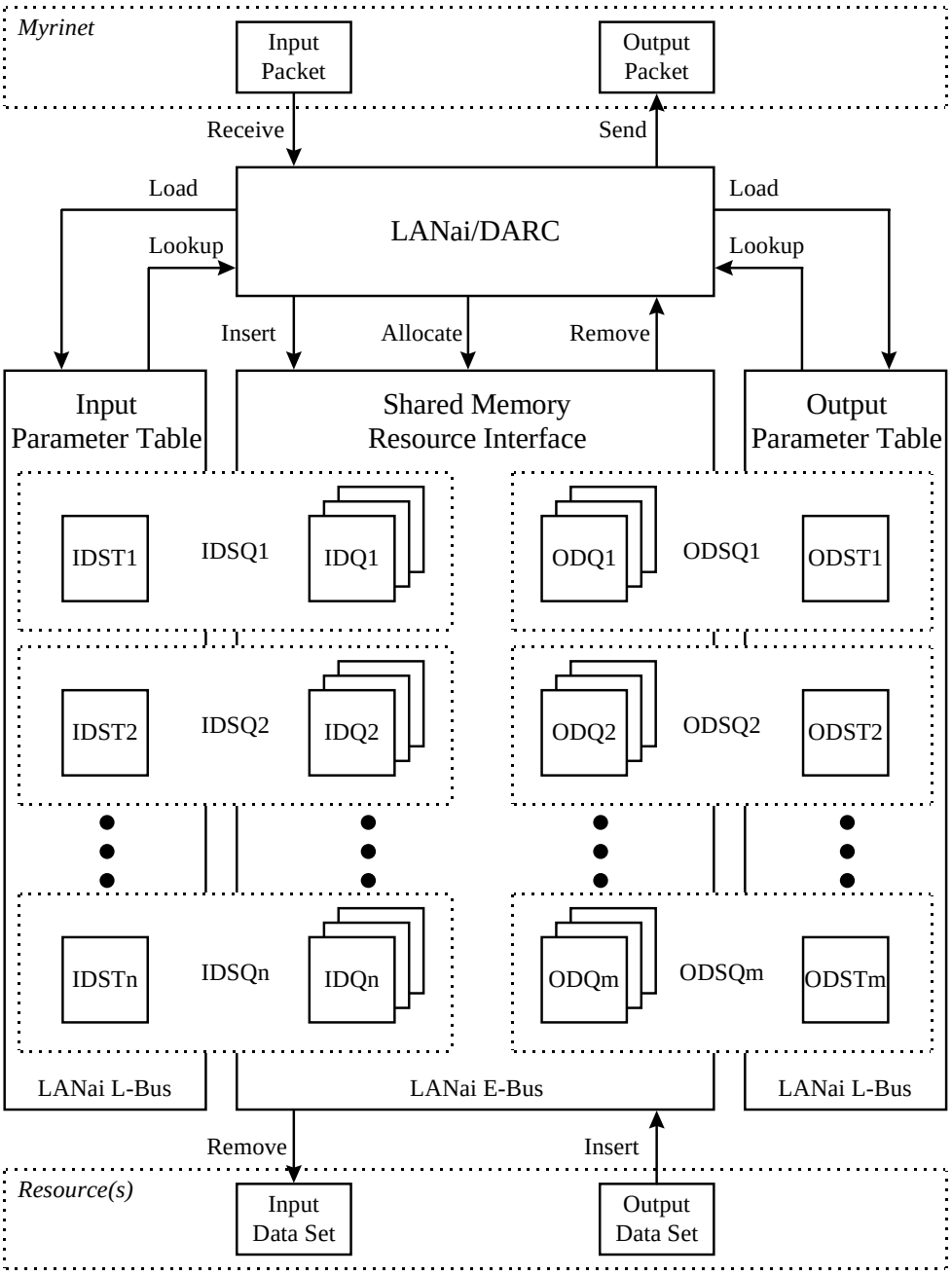



**Figure 18. DSQ Runtime Environment**

Prior to network application execution, all input and output parameter tables are loaded. This triggers the data set instantiation process where all data buffers are allocated and the queues are formed. During application execution, as data slots are received and sent, the process of notification begins. Notification requires the lookup of data set parameters associated with the current notification transaction. Depending on whether it is an input or output transaction, the appropriate input or output data structures are referenced.

For an input transaction, the input data slot is notified to its IDSQ. If a complete data set is formed by this transaction, then the data set is notified to the application executing on the resource. The data set remains queued until the application recognizes its presence and processes the data. If another input data slot arrives before the previous data set is processed, it will be buffered in the next available queue position. Error conditions that may occur during input notification include misrouted data slots, queue full conditions, and timeout conditions. A misrouted data slot or a queue full condition will result in the data slot being dropped. A timeout condition will occur if the time from the arrival of the first slot to the $n_{th}$ slot exceeds a timeout interval specified in the input parameter table. If any of these errors occur, then an error notification is sent to a designated system controller resource.

For an output transaction, an output data slot is removed from its ODSQ and notified to a remote resource. An output data slot is not notified to a remote resource until the data set that it belongs to has been completely assembled on the local resource. If several compute resources on the local node will be contributing to the output data set, then they must all indicate output data set availability before a data set will be sent slot by slot to the remote resource(s). New data sets may be constructed in the output data queues while older partially constructed sets are still in progress. When an entire output data set becomes available, it is sent one slot at a time to the remote resource(s). Any queue overflow conditions that may occur will be handled as an error notification to a designated system controller resource. The remote receivers will detect and handle any misrouted data slots or timeout conditions as part of their input transaction processing.

Figure 19 is an illustration of the scatter/gather data flow synchronization that is possible with DSQ. The network application in this example contains the following: three nodes numbered from 0 to 2, five resources numbered from 0 to 4, four input data sets numbered from 0 to 3, and four output data sets numbered from 0 to 3. Resource 0 owns the Output Data Set 0 which is partitioned into three slots. Output Data Set 0 Slot 0 is sent to Resource 1, Output Data Set 0 Slot 1 is sent to Resource 2, and Output Data Set 0 Slot 2 is sent to Resource 2. This illustrates data scattering. Data set slots are transferred between nodes as Myrinet packets. Note that node, resource, and data set numbering have global scope whereas slot numbering has local (data set) scope.

Resource 2 contains a single Input Data Set 1 that in turn contains a single slot, although it could contain multiple slots but it is simplified for the illustration. When that slot is filled, Input Data Set 1 is processed, creating the resulting Output Data Set 2. When Output Data Set 2 is available, it is sent as a single slot to Resource 5's Input Data Set 4 Slot 1. Resources 3 and 4 also contribute to Resource 5's Input Data Set 4. This illustrates data gathering. When all three slots of Input Data Set 4 are available, then that data set is processed by Resource 5.

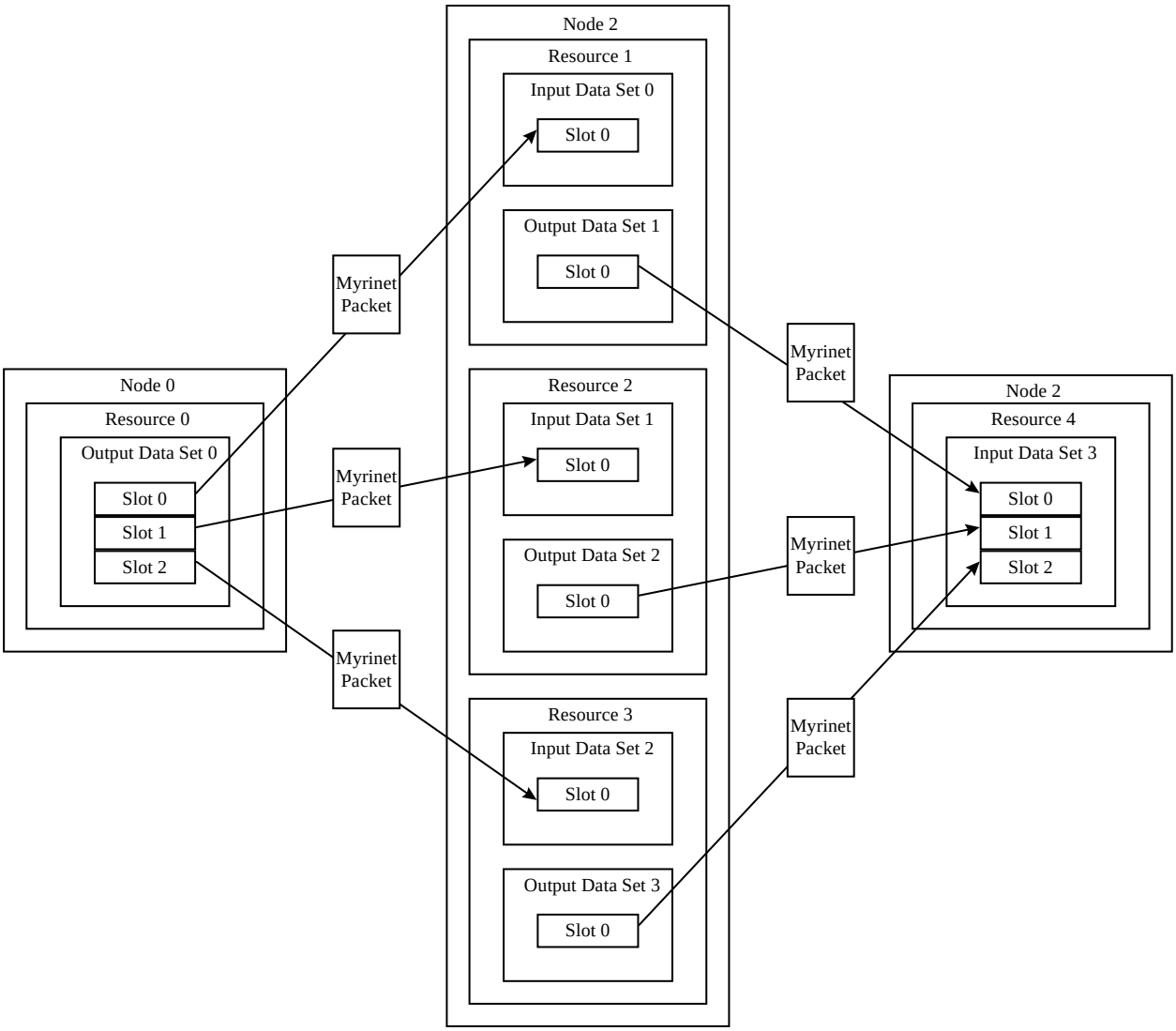



**Figure 19. Scatter/Gather Data Flow Synchronization**

Node boundaries are shown in Figure 19 because they are essential for routing Myrinet packets. EHPSCS module and chassis boundaries are not shown for simplicity, although these boundaries do affect data flow timing because of changes in Myrinet transmission protocols when crossing a chassis boundary.

#### 4.1.5.3 DSQ Performance Characterization

In the development and characterization of the DSQ paradigm, Sanders focused on the improvement of critical real-time system scalability issues. Specifically, the EHPSCS program sought to improve system program load time, initialization time, data synchronization time, compute resource efficiency, and total message passing latency over what is typically provided by commercially available systems. Sanders characterized the testbed performance running DSQ with respect to these issues.

Figure 20 shows the program load process for the testbed. Once the LANai is booted with the DARC from the node level, a one-time parameter table containing all network connectivity information is passed to it from the host. Lastly, the resource is loaded with application code. An important point is that the EHPSCS system does not use an OS on the SHARC resources, saving loading time of what would typically be 500 KB of code.

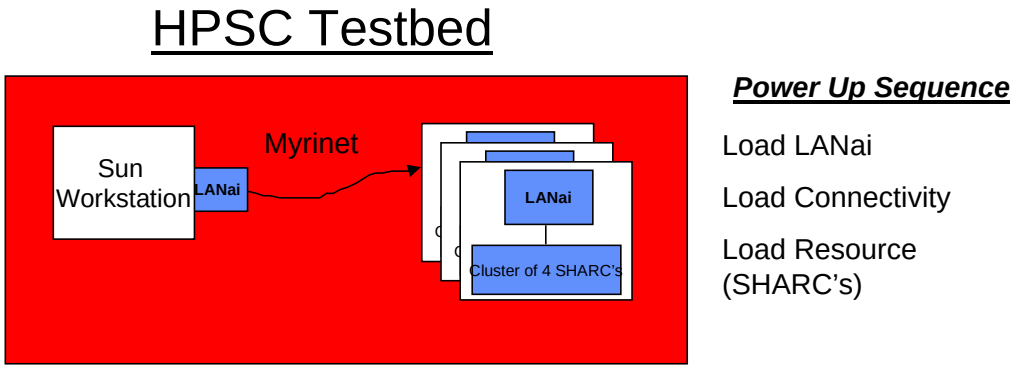
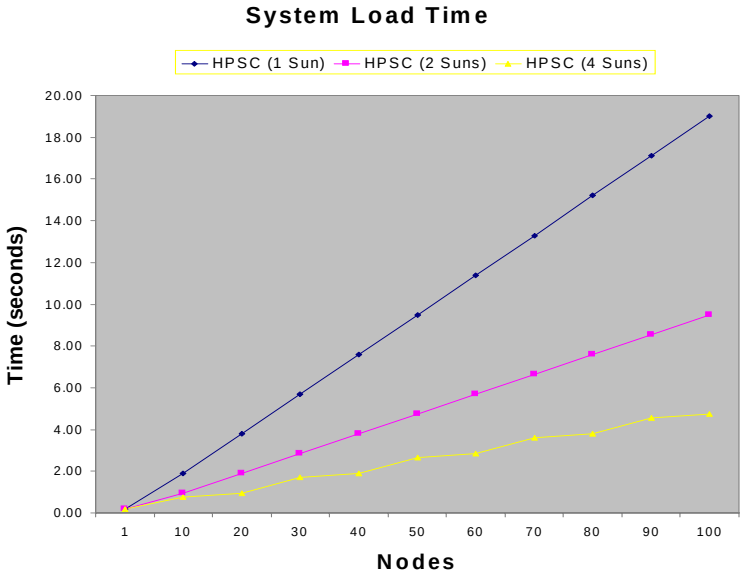



**Figure 20. EHPSCS Program Load**

The characterization data for program loading of the EHPSCS system is shown in Figure 21. The system scales in program load time with the number of nodes, with load times being proportionally reduced with the number of hosts contributing to the loading.




| Nodes | HPSC (1 SUN) | HPSC (2 SUNS) | HPSC 4 SUNS) |
|---|---|---|---|
| 1 | 0.19 | 0.19 | 0.19 |
| 10 | 1.90 | 0.95 | 0.76 |
| 20 | 3.80 | 1.90 | 0.95 |
| 30 | 5.70 | 2.85 | 1.71 |
| 40 | 7.60 | 3.80 | 1.90 |
| 50 | 9.50 | 4.75 | 2.66 |
| 60 | 11.40 | 5.70 | 2.85 |
| 70 | 13.30 | 6.65 | 3.61 |
| 80 | 15.20 | 7.60 | 3.80 |
| 90 | 17.10 | 8.55 | 4.56 |
| 100 | 19.00 | 9.50 | 4.75 |

**Program Load Consists of :** -113kbyte Executable file

**Figure 21. Program Load Characterization**

The initialization process for the EHPSCS system simply requires an attach between each LANai and resource. An attach is simply a message passed from the resource to the network interface controller. This is a fixed time of 266 us, independent of the size of the scalable system, since all attaches are done in parallel. This is significant in comparison to shared memory systems that scale in time with the number of resources because of loading and coordination of semaphores and shared memory buffer names.

Figure 22 describes the data transfer and synchronization process of DSQs on the EHPSCS testbed. The two key features in the synchronization processes are that 1.) the synchronization and data information is contained in a single packet for the EHPSCS system and 2.) notification occurs only once per data set in the EHPSCS system. The result is that the overhead and synchronization latency times for the EHPSCS system become less significant, compared to data transfer times, as packets become larger and are pipelined. In a semaphore-based synchronization system, the synchronization information and the data are transmitted separately adding a proportional amount of synchronization latency to each packet.

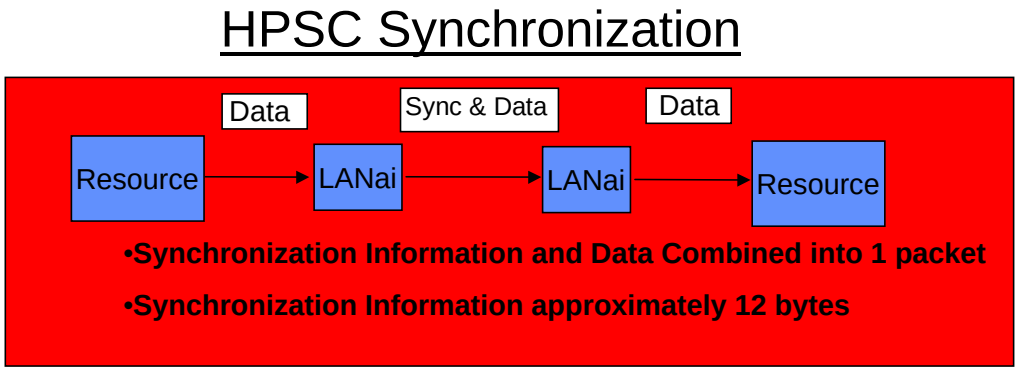



**Figure 22. Synchronization Functional Comparison**

The EHPSCS synchronization is performed in the DARC at the network level in the two-level multicomputer architecture. The resource is oblivious to the operation and is therefore more efficiently applied to application processing. The number of synchronizations does not affect the resource utilization. This also accommodates heterogeneous resource OS support. The support of different resource operating systems provides complete flexibility in hardware, software, and tool selection for resources. As resources change, the network-level software (DARC) remains.

## Total Latency Measurements (usec)

| Slot Size (bytes) | Slot Count 1 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|
| 4 | 59.68 | 87.79 | 123.92 | 196.42 | 340.94 | 630.51 |
| 16 | 60.77 | 87.76 | 123.87 | 196.38 | 342.16 | 632.78 |
| 32 | 61.10 | 88.02 | 124.39 | 198.10 | 344.40 | 637.29 |
| 64 | 60.93 | 88.39 | 125.92 | 200.06 | 348.20 | 645.59 |
| 128 | 61.49 | 89.92 | 127.82 | 203.80 | 355.20 | 662.22 |
| 256 | 62.21 | 92.51 | 132.31 | 211.88 | 372.00 | 690.71 |
| 512 | 63.65 | 96.79 | 140.21 | 227.79 | 401.77 | 749.79 |
| 1024 | 67.82 | 109.08 | 166.74 | 280.67 | 508.55 | 967.93 |
| 2048 | 82.44 | 144.21 | 226.33 | 390.80 | 721.13 | 1380.06 |
| 4096 | 118.05 | 219.65 | 352.71 | 620.28 | 1153.40 | 2222.01 |
| 8192 | 195.49 | 371.42 | 608.17 | 1079.52 | 2024.64 | NA |
| 16384 | 347.96 | 680.23 | 1121.05 | 2004.27 | NA | NA |
| 32768 | 658.40 | 1298.09 | 2152.14 | NA | NA | NA |

| DSQ Overhead | | | | | |
|---|---|---|---|---|---|
| bytes | 4 | 4096 | 8096 | 16384 | 32768 |
| Time usec | 51 | 40 | 30 | 26 | 18 |

**Figure 23. DSQ Total Data Transfer and Synchronization Latency Measurements**

**Figure 23** and Figure 24 show the Total Data Transfer and Synchronization Latency Measurements for the DSQ running on the APU board. The measurements varied over data slot size and data slot per data set count. It should be noted that the worst-case overhead latency was measured at 51 us in the simplest data set case. As the data set becomes more complex, the overhead latency decreases.

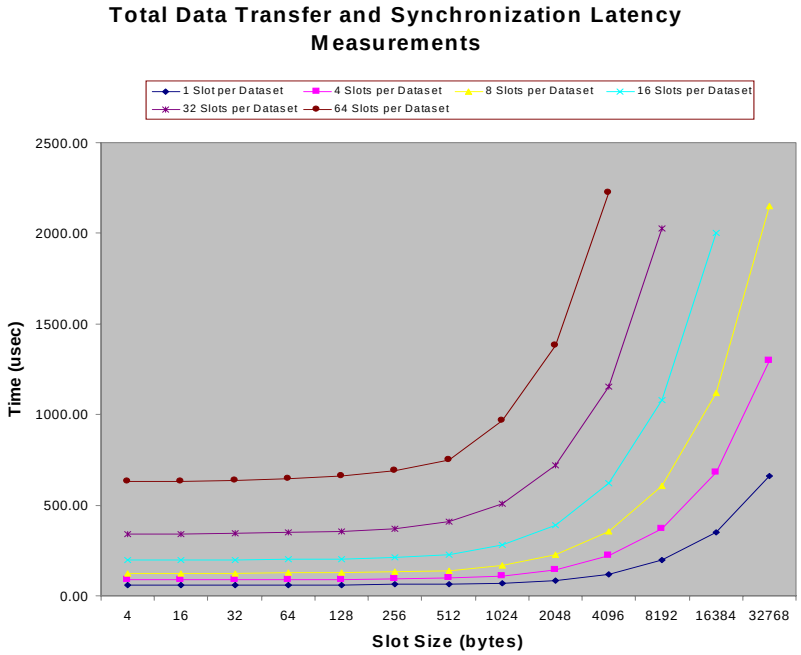



**Figure 24. DSQ Total Data Transfer and Synchronization Latency Measurements**

The EHPSCS development recognizes that system complexity is in the data synchronization and is therefore packaged this away from the compute resources for network communication efficiency and performance gains, approaching the bandwidth of the network. A user should not have to compromise system flexibility or scalability to reduce communication latency.

### 4.1.6 Testbed Benchmark Application Demonstration

A first-generation EHPSCS hardware/software functional testbed using discrete components was completed in June 1996. The EHPSCS functional testbed provides up to 11 GFLOPS of peak performance in a single VME chassis using 11 Arithmetic Processing Unit (APU) modules. The first demonstration of the EHPSCS testbed along with an application demonstration was completed in late June, 1996. The demo consisted of the processing-intensive Householder matrix transformation function that is part of the Space-Time Adaptive Processing (STAP) algorithm. The Householder function was allocated to a parallel SHARC processor node on an APU module. Real-time sensor data sourcing was simulated by a second node with data routed through the Myrinet switch network. The successful completion of the application demonstration validated the EHPSCS functional testbed hardware/software, the software development tools and environment, and the basic two-level multi-computer architecture concept.

### 4.1.7 Message Passing Interface Development

Another major goal of the architecture was software scalability and portability. By nature of the two-level multicomputer, all application code is decoupled from the network communication code. By developing application code in a standard Application Programming Interface (API), Message Passing Interface (MPI) for instance, the application code becomes portable from platform to platform. The obvious advantage of this is that software can be developed in a non-real-time workstation environment and seamlessly ported to real-time target hardware – in this case, the EHPSCS Multicomputer Testbed.

An implementation of MPI was incorporated as part of the EHPSCS multicomputer testbed development in collaboration with the ACP program. MPI is a standard application programming interface for writing portable and scalable high-performance message passing based parallel programs in a heterogeneous system. It is highly conducive to performance-oriented parallel applications and contains a broad scope of functionality. In addition, MPI is becoming widely accepted in the parallel computing community. The MPI standards forum is representative of government, industry and academia. The final MPI-1 standard was published in May 1994 and the MPI-2 standard in July 1997. Sanders is a member of the MPI-RT forum. The purpose of the MPI-RT forum is to provide extensions to MPI for real-time systems and resource-constrained systems. Sanders has been a driving force in the MPI-RT forum to define the resource constraint (RC) standard. The target date for the MPI-RT standard is summer 1998.

The critical design requirements governing the MPI implementation were resource memory constraints, low latency, and modularity to allow retargeting of code. The implementation developed is based on the MPI-2 real-time subcommittee definition of an embedded version of MPI-1.1. The MPI implementation structure consists of three layers; API, Protocol, and Network. The API provides a portable interface for writing parallel applications. The Protocol layer manages MPI specific protocols along with flow control and reliability. The Network layer

provides MPI packet transport functionality. It also uses a Myricom mapping utility for dynamic network mapping and EHPSCS-developed software tools for loading and administration.

This modular implementation effectively allows retargeting of code to different architectures in a limited development time. It also allows for retargeting code to different locations within an architecture; for example, limited resources in embedded targets. Layers were placed in the appropriate part of the communication hierarchy on a target-specific basis. For a SUN workstation, only the low-level network send/receive functionality is placed on the LANai. The EHPSCS APU has enough SRAM to support MPI implementations because most of the MPI protocol is pushed to the LANai. For an optical processor application, all the MPI protocol is pushed into the LANai. Figure 25 shows an implementation specific layering.

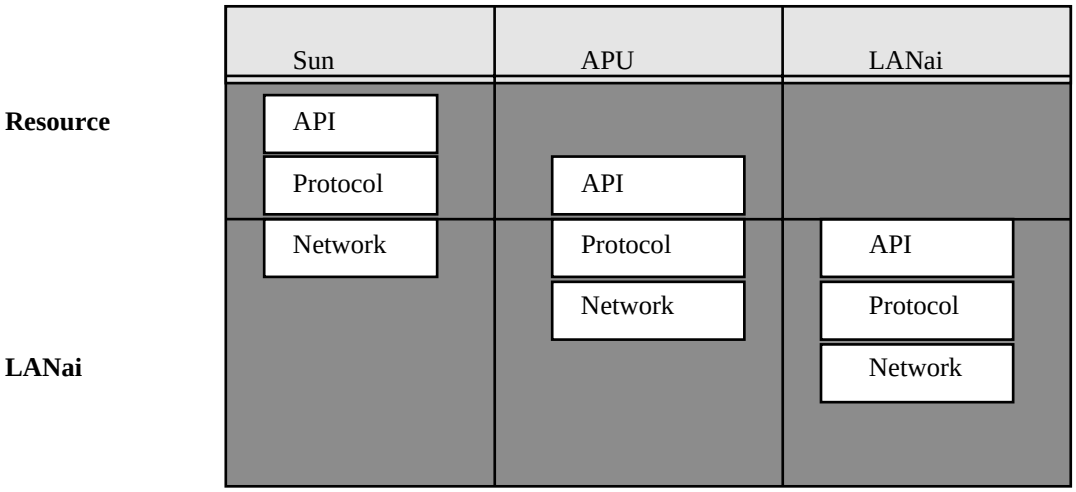



**Figure 25. Implementation-specific Layering**

In conjunction with the MPI implementation, an environmental toolset was developed. These tools are specified in the EHPSCS MPI Users Reference Guide, section 7.3, and include MPICC, MPIRUN, MPISETUP, MPINODES, and a SHARC "printf".

MPICC provides a simple method of compiling MPI programs. Compilation is performed with the appropriate MPI flags and links to appropriate libraries. MPICC contains options that allow cross compile for the SHARC and PPC, links to alog profiling libraries, and enabling of SUN debug execution under gdb or ddd control.

MPIRUN provides a convenient method to run MPI programs. MPIRUN contains options that allow the user to specify the number of nodes to run, specify the nodes and executables in a process file, generate a script file instead of running, or run a profiling tool upon program termination.

MPISETUP and MPINODES provide the user a seamless method of determining the topology of the network. MPISETUP, in conjunction with a Myricom Myrinet Mapper utility determines the available nodes and associated routes. In a static network configuration, MPISETUP need only be run once. MPINODES provides a list of available network nodes.

A SHARC “printf” was created to ease application development as well as for MPI implementation debug. It uses functionality in the MPI library to send printf information to the SUN for display.

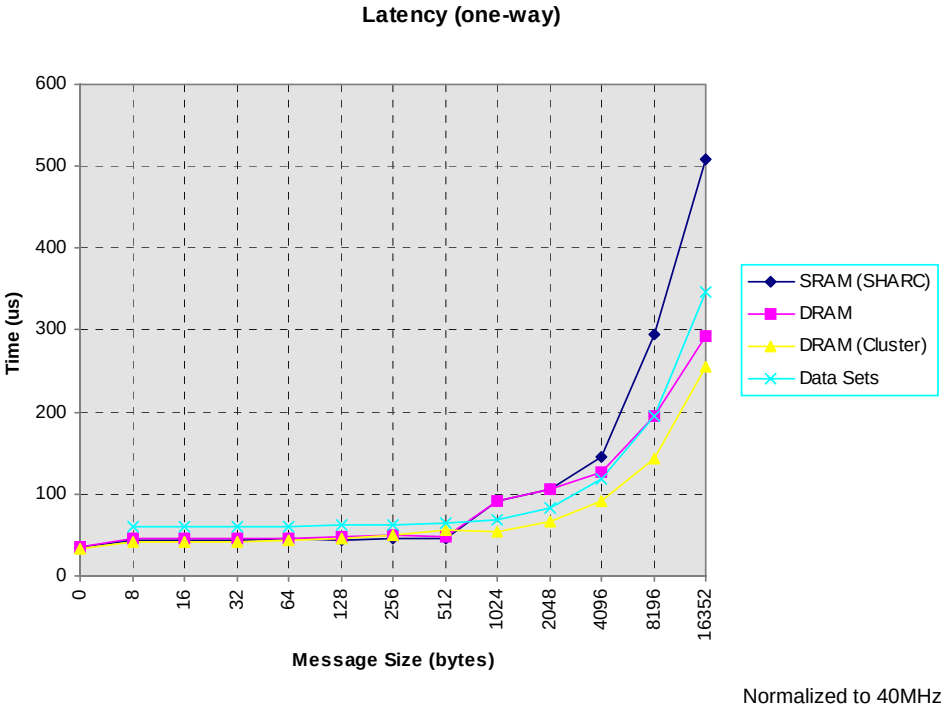



**Figure 26. MPI Latency Characterization**

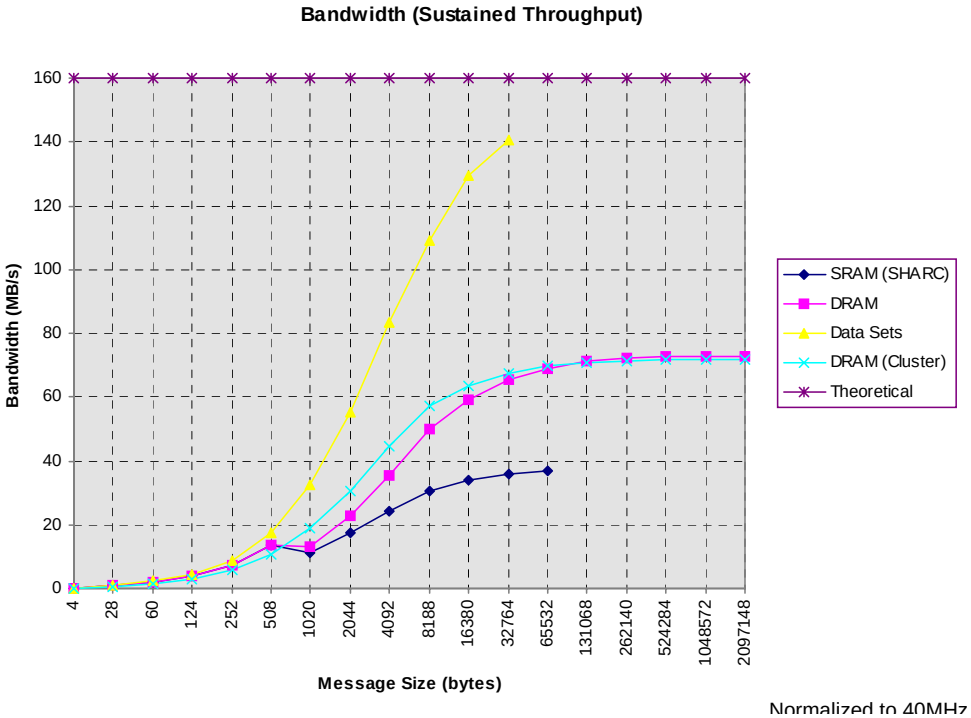



**Figure 27. MPI Bandwidth Characterization**

Figure 26. MPI Latency Characterization and Figure 27. MPI Bandwidth Characterization show the characterization results using MPI on the EHPSCS testbed. The Sanders MPI implementation provided the proof of concept required to transition EHPSCS technology to commercial facets. MPI was successfully ported to a "dumb" node and to a two-level multicomputer consisting of a "smart" node SUN and ADI SHARC based resource constrained nodes. Generic application porting was relatively seamless. Peak bandwidth for large messages is good (122 MB/s – 76% of peak 160 MB/s bandwidth). In addition, MPI applications that can overlap communication and computation perform very well.

However, the implementation does have some shortcomings. End-to-end latency numbers are somewhat poor (35 us for 0 Byte messages). This is because the control program is resident on the LANai, which is a performance-limited processor. A faster communication processor is necessary for improved latency. In addition, the Sanders MPI implementation may not be interoperable with other MPI implementations. MPI is an API specification, not a protocol specification. Vendor implementation of MPI is protocol specific.

### 4.1.8 Advanced Packaging

The EHPSCS architecture is supported by a scalable packaging approach to produce flexible and cost-effective embedded mission application solutions. For avionics applications for instance, an implementation is needed that minimizes size, weight, and power (SWAP). In addition to a standard printed wiring board (PWB), the project has developed advanced scalable packaging solutions using High Density Interconnect (HDI) MultiChip Module (MCM) process technology from General Electric Corporate Research and Development (GE CRD).

By their very nature, MCMs reduce size, weight, and power over an equivalent discrete representation. Size and weight are reduced by the elimination of packaging and PWB board space. Power is reduced due to the fact that components are driving low-capacitance MCM traces and die pads rather than PWB traces and package pins. But SWAP reduction comes at a cost. MCMs have traditionally been very expensive, both in terms of cost/risk and development time. Most integrated circuit vendor production lines are optimized for packaged part production, making the cost of handling and testing unpackaged die high. MCMs also require a development cycle of approximately 6-9 months, which can impact rapid insertion opportunities of state of the art electronics. The advanced packaging effort under the EHPSCS project addresses both SWAP reduction for constrained military insertions and improved cost and risk of MCM applications for EHPSCS users.

Three generations of MCMs were developed for the EHPSCS program. The Embedded DSP MCM was functionally equivalent to one-half of an APU board and was a 5V design implemented in standard HDI technology. The DRAM MCM was implemented in MCM-Flex technology in an attempt to reduce the cost of the module. And lastly, the Low Power DSP MCM again was functionally equivalent to one-half of an APU board but was a 3.3V design implemented in a modular fashion in MCM-Flex technology. These are described in detail in the paragraphs that follow.

#### 4.1.8.1 Embedded DSP MultiChip Module Development

The purpose of the 5V Embedded DSP MCM was to demonstrate a modular packaging architecture that is consistent with the EHPSCS design with a reduction in SWAP for use in highly constrained applications. The DSP MCM consisted of a single processing node, virtually identical in functionality to each node on the APU board, four ADSP-21060 SHARC DSPs, 1 MB of SRAM, a LANai network interface, and an eight port crossbar switch. The block diagram of the DSP MCM is shown in Figure 28.

This advanced packaging exercise produced a DSP processing node with 480 peak MFLOPS in a 3.2" x 2.5" package, providing an impressive 60 MFLOPS/in$^2$ processing density. This is an increase of a factor of approximately 2.5 over the discrete APU implementation. The 5V DSP MCM validated the node architecture packaging concept and provided an embedded system transition path to insertion program users.

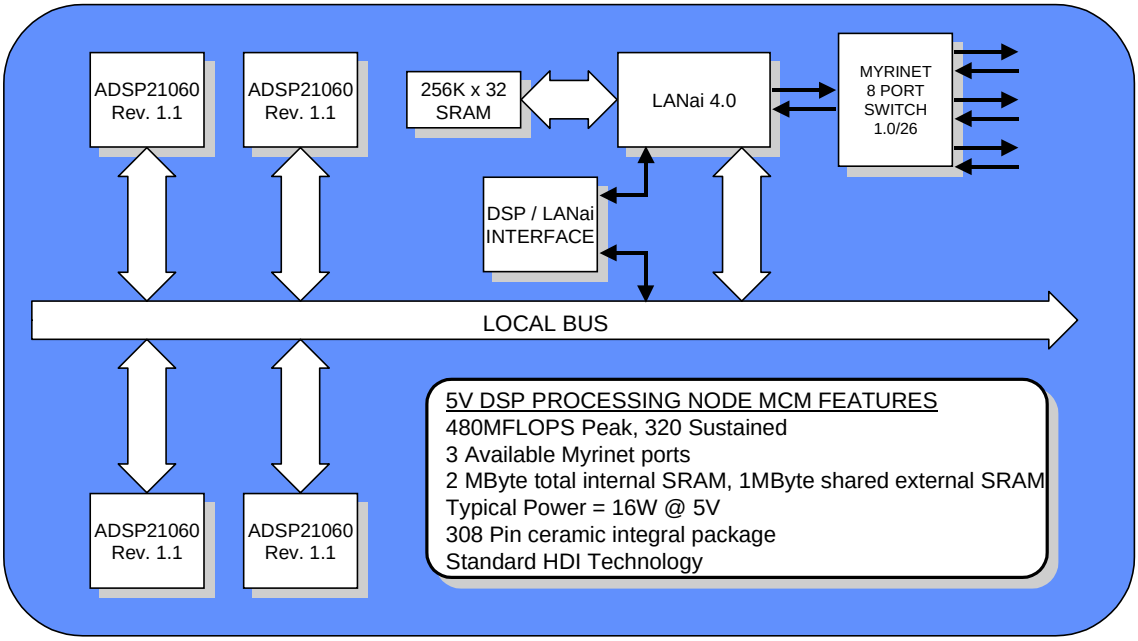



**Figure 28. 5V Embedded DSP Multichip Module**

This MCM was designed concurrently with the APU Rev. 0 board. The APU served as a discrete prototype for the MCM before the MCM was actually fabricated, which allowed some design changes to be incorporated into the MCM before actual fabrication.

This MCM was developed in baseline HDI technology. A side view of this technology is shown in Figure 29. This technology requires milling on the package for individual die cavities in the package to achieve planarity across top surface of die after which layers of multilamination are applied for interconnect. A pictorial description of this process is attached at the end of this document.

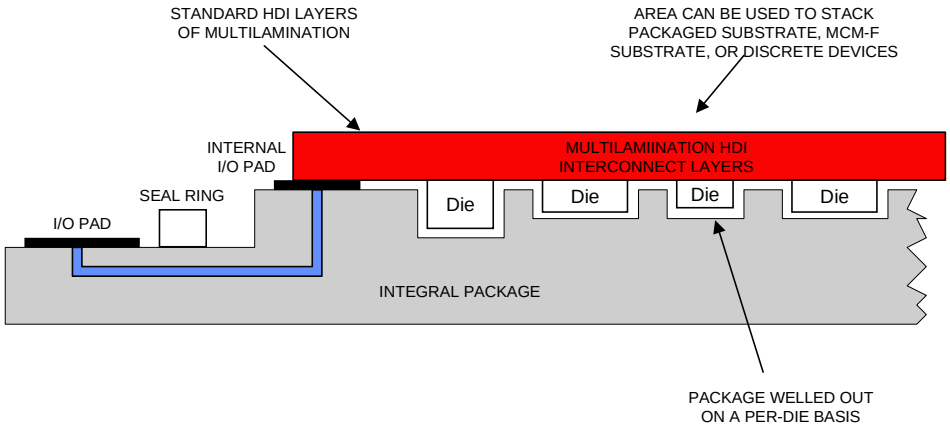



**Figure 29. EHPSCS 5V Embedded DSP MCM – Baseline HDI side view**

HDI technology offers many unique features over other MCM technologies that were used in this particular implementation. One feature is the ability to attach packaged parts or die directly to the top of the interconnect layers. Another feature is the reparability of the interconnect. Due to great demand from the personal computer market for fast SRAM at the time of this development, it was impossible to acquire the necessary memory for this project in die form. Therefore, packaged SRAM were soldered directly onto the HDI layer in place of die. In addition, for risk mitigation reasons, the 3 one-time-programmable PALs used for the SHARC/LANai interface were buried in the MCM like all the other die, but also had pads brought up to the top layer (with the packaged SRAM). In the event that the original contents of the PALs needed to be changed for design functionality, the traces to the buried PALs could be cut, and a new PAL die could be wirebonded to the top layer. As it turned out, all three PALs did change and new ones were bonded to the HDI. One of the changes required a trace cut with two jumper wires. Both of these workarounds were critical to the success of demonstrating the embedded MCM. The reparability of these modules by GE CRD was impressive. Photos of the depelted and reworked MCMs are shown in Figure 30.

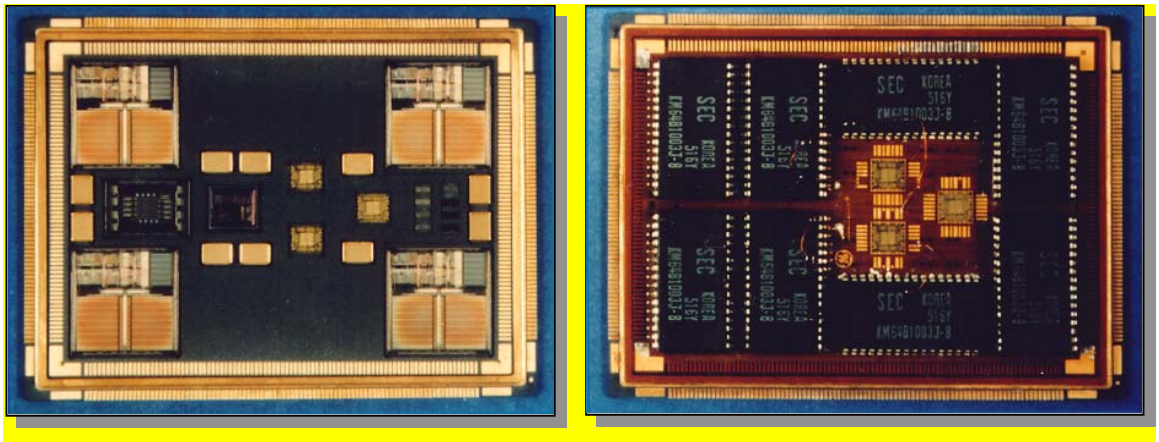


**Figure 30. EHPSCS 5V Embedded DSP MCM**
**Left: Depelted MCM (no HDI interconnect)**
**Right: Complete MCM with discrete components**

Fabrication was completed and testing began on three 5V Embedded DSP MCMs in early 1996. The modules were tested standalone at GE CRD with confidence tests based on a version used to test the APU modules. The internal node functionality was verified standalone at 25 MHz. In all, three MCM units were produced under this prototype effort.

### 4.1.8.2 DRAM MultiChip Module Development

The 5V DRAM MCM development objective was consistent with the 5V DSP MCM and included investigation into reduction of the cost of MCM production by using 'plastic HDI' or MCM-Flex (MCM-F) technology. A great deal of the cost of standard HDI technology is in the series of precision steps necessary to produce a module; the milling of a package on a per-die type basis and the multilamination process, for instance. MCM-Flex technology is able to use less expensive interconnect than baseline HDI multilamination. In addition, in the MCM-F process, the die are bonded directly to the interconnect rather than the interconnect being bonded to the die as is the case with baseline HDI. This eliminates the need for costly precision milling in a package to attain planarity across the die surfaces. The DRAM MCM exploited those features to demonstrate a lower-cost module while retaining the silicon density of standard HDI technology. A pictorial description of this process is attached at the end of this document. For more information on the MCM-F process and this MCM in particular, please refer to http://www.crd.ge.com/csetl/edci/projects/cof/index.html.

This advanced packaging exercise produced a 32 Mbyte asynchronous DRAM MCM. The module features 0-wait state burst accesses, in-circuit reprogrammability, and parity error detection. The block diagram is shown in Figure 31.

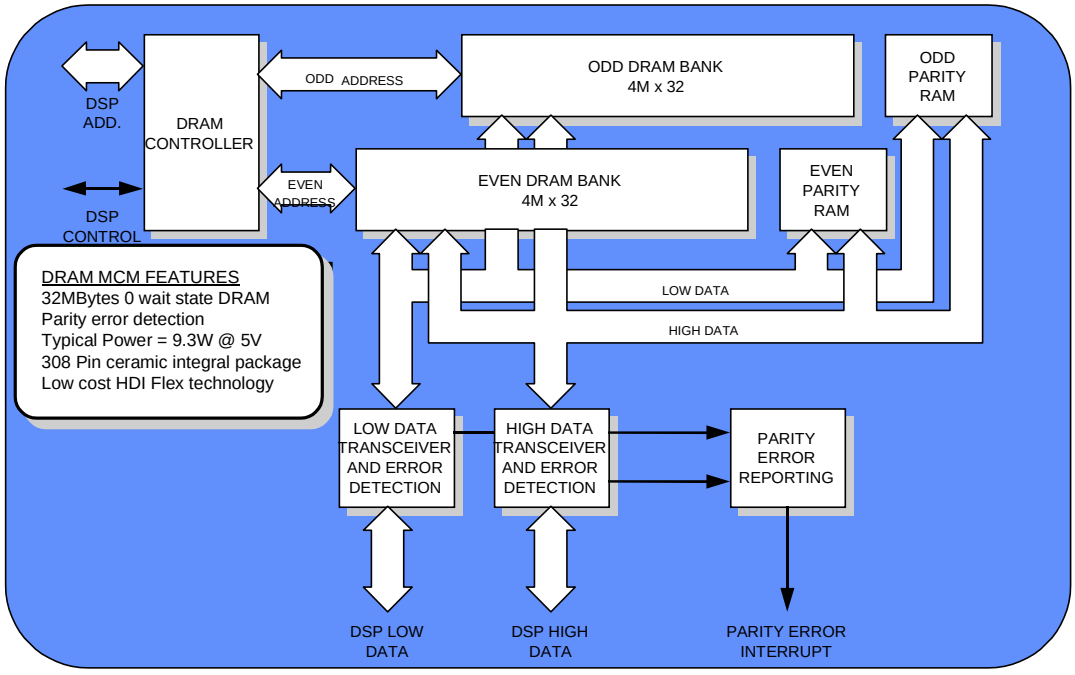



**Figure 31. EHPSCS DRAM Multichip Module Block Diagram**

The interconnect used was a double-sided, prepatterned flex with one spin-on layer. The prepatterned flex is available commercially, while the spin-on layer is a less expensive interconnect process relative to one layer of multilamination of standard HDI. The monolithic plastic module was then wirebonded into the same 3.2" x 2.5" ceramic package used for the 5V embedded DSP MCM. The resulting footprint is 2.2 times smaller in board area than the equivalent discrete circuitry. The DRAM MCM did not require the package milling necessary in standard HDI technology, thus reducing processing steps and cost. A side view of this MCM-F technology is shown in

Figure **32**. This module development added low-cost technology to the architecture packaging concept and supplemented the embedded system transition path to insertion program users.

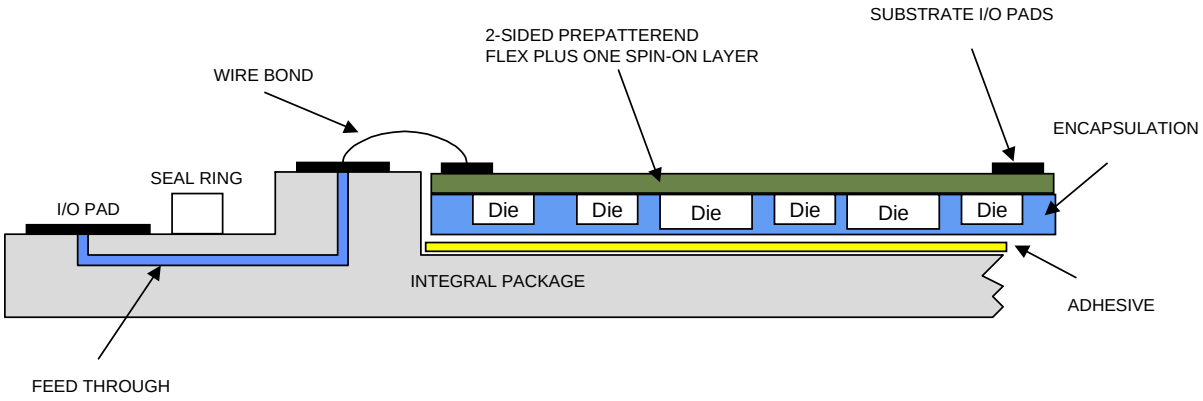



**Figure 32. EHPSCS MCM/Flex Technology**

Two MCM substrates were fabricated at the GE CRD facility, one of which was tested standalone on an IMS tester at GE CRD in June 1996. The DRAM controller design is in-circuit programmable allowing design changes to be made after the DRAM MCMs were fabricated. The DRAM MCMs were tested for functional operation based on a 5-cycle latency at 1 MHz. The operating speed was limited by IMS tester setup. The DRAM functional tests provided verification of the physical design using MCM-F. A photo of the MCM is shown in Figure 33.

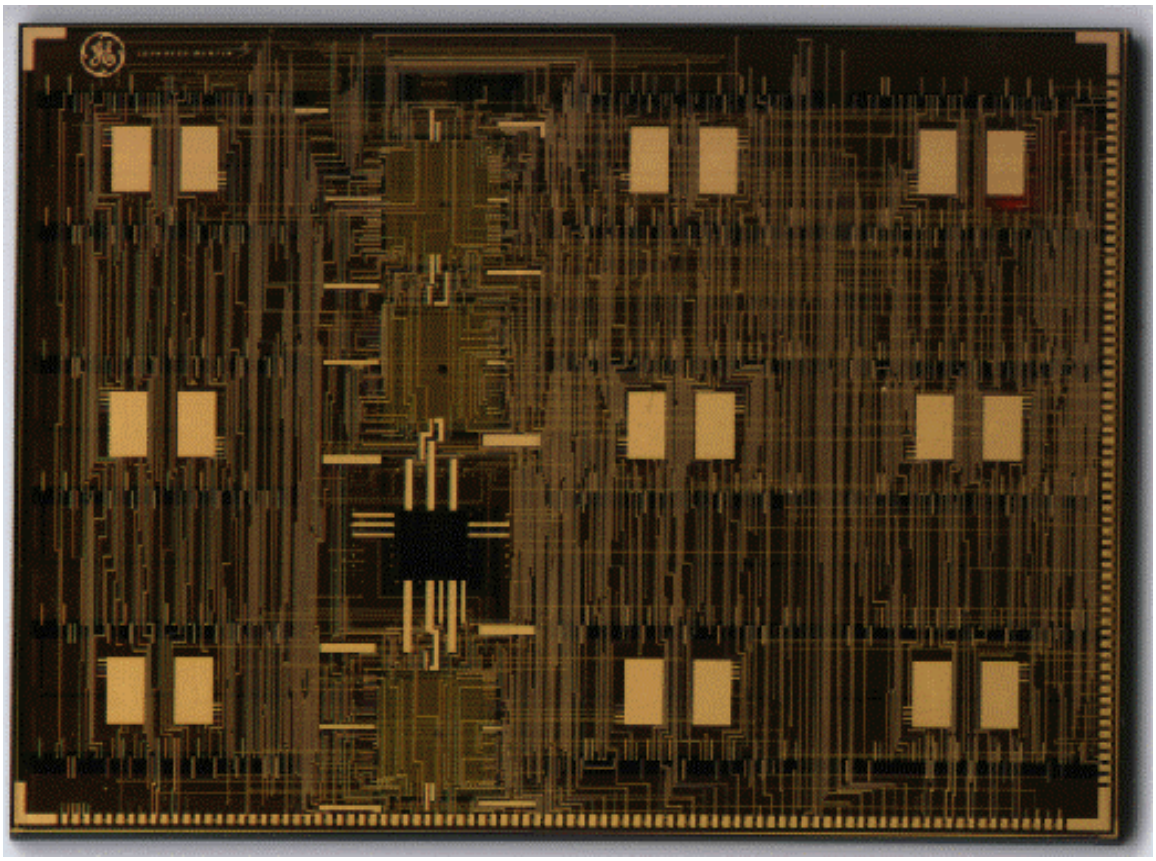


**Figure 33. EHPSCS DRAM Multichip Module Photo**

This development successfully demonstrated DRAM functionality as well as MCM cost reduction. In addition, the entire DRAM MCM development provided an important lesson learned. Commercial DRAM products increase in density by a factor of four every 18-24 months. This density increase must be compared to the SWAP savings, development cost, and schedule of an MCM development. The DRAM MCM development experience produced data points indicating that constrained applications must be justified by cost and schedule for SWAP savings compared to similar savings provided by COTS DRAM product cycle advancements. This experience was rolled into the development of the Low Power DSP MCM.

#### 4.1.8.3 Low Power DSP MultiChip Module Development

The objectives of this effort were to reduce power and to investigate approaches to reduce cost and increase MCM reuse. This effort began with a proven functional design from the 5V DSP MCM and discrete APU experience. Power reduction came in the form of transitioning the 5V DSP MCM to this 3.3V version. Two process steps were taken to accomplish the other two goals. First, the MCM-F process was used to minimize the interconnect cost. Secondly, the design was broken up into three modular tiles.

The tile concept served several purposes. By breaking down the whole node design into modular tiles, the routing complexity of each tile was much simpler than a monolithic MCM and could be routed in fewer layers. Eliminating the need for spin-on dielectrics or multilamination and

routing on a lone double-sided prepatterned flex layer is the lowest cost HDI interconnect. In addition, the smaller tiles should promote tile-based testing to avoid the high costs of known-good-die (KGD). Lastly, the modular concept will enhance design reuse in other developments. A side view of the tile-based MCM-F technology is shown in Figure 34.

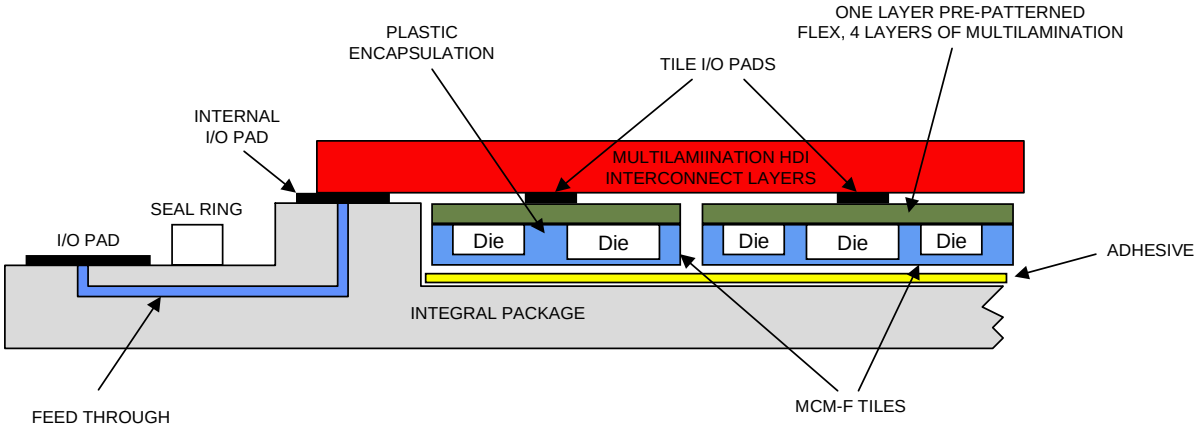



**Figure 34. Tile-based MCM-F Technology Side View**

The block diagram of the Low Power DSP MCM is shown in Figure 35. It is functionally identical to the 5V version completed in 1996, but excluded the Myrinet crossbar switch to reflect a consistent EHPSCS modular node partition. By nature of the 3.3V version of the design, power density is reduced by at least 50%. It was divided up into three tiles as shown in the diagram. It has features similar to the 5V DSP MCM, but with a typical power dissipation of 7.5W.

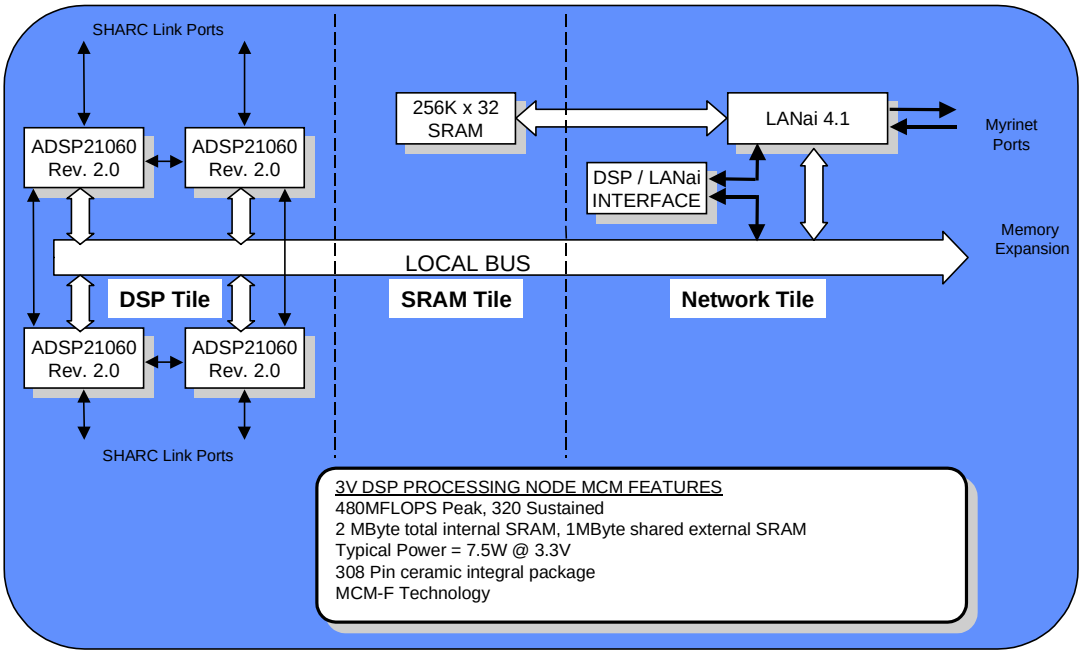



**Figure 35. EHPSCS 3V DSP Processing Node Functional Block Diagram**

During the layout phase of the development, it was discovered that even by dividing the design into three modular tiles, the routing density of each still surpassed that supported by the low-cost single layer of double-sided prepatterned flex interconnect. Additional layers of interconnect were necessary on each tile. The final interconnect for each tile consisted of a layer of single-sided prepatterned flex and four layers of multilamination. In addition, four layers of multilamination were required to interconnect the tiles with each other and the I/O pins of the package. In terms of fabrication costs, the investigation has concluded that the fabrication cost was increased compared to the 5V DSP MCM due to the need for eight total layers of multilamination. Only five layers of multilamination were used for the complete 5V Embedded DSP MCM.

The unanticipated routing complexity on the tiles combined with the learning curve involved in the tile design resulted in a schedule delay of five months and caused the elimination of MCM tile testing step as originally planned.

The first lot of MCMs yielded five testable modules in May 1997. A sixth module experienced irreparable delamination between layers during manufacturing and was not completed. A photo of the depelted MCM, displaying the 3 tiles, is shown in Figure 36. As testing began, all five MCMs exhibited similar gross failures. This development did not yield a functional prototype due to potential process defects and/or bad die in the fabrication and assembly of the MCM prototype. Complete results and analysis are found in Test Results and Fault Analysis For High Performance Scaleable Computing 3.3V Digital Signal Processor Multi-Chip Module, Revision - , which is an attachment to this report.

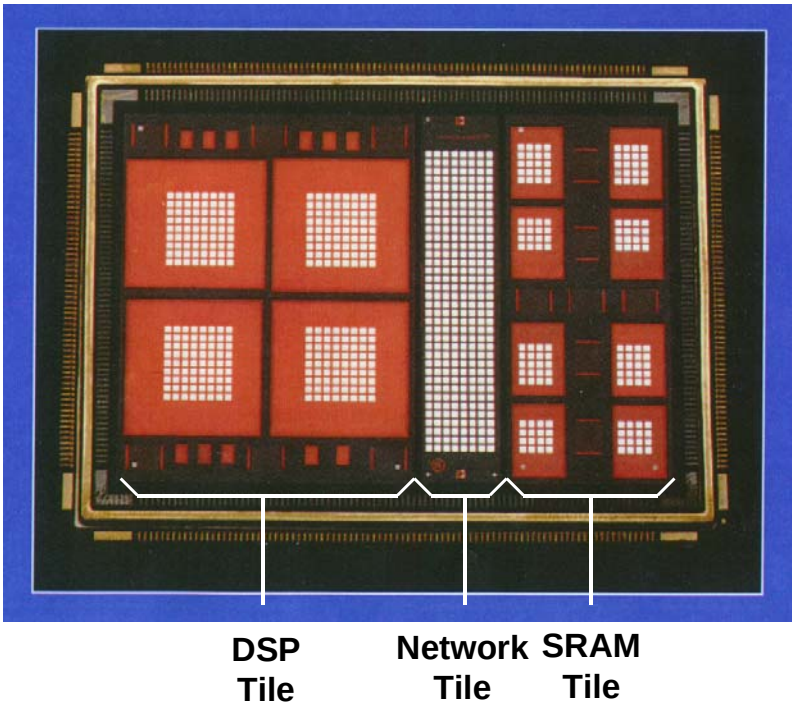



**Figure 36. 3V DSP Processing Node MCM**

The UUV insertion program utilized the 3.3V DSP node design in discrete form and successfully demonstrated its functionality and power savings (over 5V version) of at least 50% in November 1997.

This investigation concluded that the tile-based process, based on current double-sided prepatterned flex interconnect technology, is limited for complex digital designs. This has resulted in the need for multilamination processing steps and higher interconnect cost in our design. The cost data exists now to trade off the cost of a tiled design versus the cost of KGD. Lastly, die product cycles (die size/process changes) may be too short to enable effective reuse of tiles and may greatly limit the lifespan of a module design.

### 4.1.9 Technology Transition and Insertion

One of the goals of the EHPSCS program was to enhance the High Performance Computing technology base. That goal was successfully accomplished when the first insertion testbed was delivered to the STAP program out of Lockheed Martin in Syracuse, NY in July, 1996. Since that date, Sanders has delivered 14 testbeds which include 70 APUs and 18 MTEMs to seven different insertion programs, as listed in Table 1.

| Insertion Program | APU Deliveries | MTEM Deliveries | Chassis Deliveries |
|---|---|---|---|
| | | | |
| ACP | 29 | 9 | 9 |
| STAP | 20 | 4 | 2 |
| P507 | 4 | 1 | 1 |
| UYS2A | 2 | | |
| UUV | 1 | 1 | |
| Others (Internal) | 14 | 3 | 2 |
| | | | |
| TOTALS | 70 | 18 | 14 |

**Table 1. EHPSCS Technology Insertion Programs**

The insertion programs cover a wide range of applications, some of which are briefly described in the sections that follow.

In addition to these technology insertion programs, the EHPSCS technology was also licensed to a commercial vendor of signal processing systems, CSPI. In September 1997, Sanders and CSPI completed a technology licensing agreement for the design of the EHPSCS APU hardware and the EHPSCS testbed system software to CSPI. CSPI, using this technology, has announced products providing up to 16 Analog Devices SHARC DSPs in a single VME 6U slot based on the EHPSCS technology. Designs incorporating this licensed technology will provide heterogeneous high-performance multicomputer solutions with leading COTS price performance. SHARC products based on this technology will be used to further expand CSPI's MAP 2000 Series High-Performance MultiComputer product line. Through this licensing agreement, the EHPSCS program has been successful in meeting the goal of inserting the technology into the High Performance Computing commercial community.

#### 4.1.9.1 STAP Insertion Program

The Space Time Adaptive Processing (STAP) Program is a DARPA-sponsored effort based in Lockheed Martin Ocean, Radar, and Sensor Systems in Syracuse, NY. The program seeks to demonstrate and validate the EHPSCS technology for next-generation Airborne Early Warning radar. The program inserted two EHPSCS testbeds, one adaptive array processor and one signal processor, into an air-based radar testbed. The adaptive array processor requires some very low latency communications in the beamformer and therefore takes advantage of the SHARClink connectivity between APUs in the testbed to enhance the Myrinet for this communication path. The STAP radar successfully demonstrated clutter and jamming suppression and real-time rooftop operation in May 1997.

#### 4.1.9.2 AN/UYS-2A Insertion Program

The AN/UYS-2A upgrade is a Lockheed Martin Advanced Technology Labs effort and is jointly sponsored by the U.S. Navy, the Defense Research Projects Agency's (DARPA) Rapid Prototyping of Application-Specific Signal Processors Program Office, and DARPA's High Performance Scaleable Computing (HPSC) Program Office.

The goal of the upgrade is to demonstrate a 15X processing-performance improvement in the AN/UYS-2A - the Navy's standard signal processor - over existing implementations at one-third the schedule and cost of the original development. Features include the implementation of a 2-GFlop Floating Point Commercial Arithmetic Processor (FPCAP) on a Standard Electronic Module-format E (SEM-E) module, which is set in the AN/UYS-2A. The EHPSCS APU hardware and software is the basis of the FPCAP design. The UYS2A processor upgrade FPCAP inserts into the Arithmetic Processor slots (three SEM-E) while retaining all hardware and software interfaces to the existing UYS2A Enhanced Modular Signal Processor. The processor is used in the Airborne Low Frequency Sonar (ALFS) for low-frequency sonar active processing. The order-of-magnitude increase in processing density allows the implementation of a new class of tracking/correlation algorithms without increasing chassis size or power systems.

The UYS2A program also was responsible for integration of the SPOX-MP operating system on to the EHPSCS testbed. SPOX-MP provides a processor-independent application program interface for multiprocessor programs requiring real-time operating system services. Typical single-processor kernel services provided by SPOX-MP include multitasking, intertask synchronization (semaphores), intertask communication (queues and mailboxes), and device-independent I/O interfaces (streams). SPOX-MP extends these services across multiple processors for multiprocessor semaphores, mailboxes, and streams.

#### 4.1.9.3 ACP Insertion Program

The Advanced Common Processor (ACP) program is a classified research and development program that integrates a number of emerging technology initiatives aimed at providing an order-of-magnitude improvement and a cost-effective compute solution for a variety of embedded high performance scalable computing applications. The ACP program provided synergistic research and development with the EHPSCS program in the development of MPI and the MTEM module. The ACP program demonstrates the methodologies required to ensure that evolutionary

processing technologies are available for low-cost insertion into today's and tomorrow's ground-, air-, and space-based embedded signal processing systems. Through the common testbed platform, the ACP program focused on technology insertions for an expanded set of user applications, including leading algorithms in image exploitation, optical technology, and network bridging. There are currently nine testbeds in various ACP sites with several more on order for the purpose of operational processing later in 1998.

#### 4.1.9.4 UUV Insertion Program

The Unmanned Underwater Vehicle (UUV) program is a DARPA-sponsored research program within the Signal Processing Algorithms and Application (SPA&A) Directorate of Sanders. The objective of the UUV program is to implement a scalable high-performance embedded signal processor for UUV applications such as 3D imaging for forward-looking sonar, side scan sonar, acoustic comms, and navigation. The program employed the EHPSCS testbed design for the development of two board types (APU and the Interface Unit) for insertion into a custom form-factor, torpedo-like shell. UUV demonstrated the system in 1998 and is in the process of final integration in the custom form factor.

#### 4.1.9.5 P507 Insertion Program

P507 is a classified space application program at Lockheed Martin Astronautics in Denver, Colorado. The program has developed hardware to channelize fast and wide A/D converter data into the testbed via Myrinet for processing. A significant amount of work has been done in virtual prototyping. This resulted in a software testbed that was developed using MatLab, Netsyn, and GEDAE where functional software testbed models are validated on the hardware testbed. The program has also developed a “suitcase” testbed to support a single APU module as a low-cost, portable application development platform alternative to the EHPSCS chassis.

### 4.2 Software Tools

Under the EHPSCS program, Sanders has been successful in leveraging multiple ongoing DARPA-sponsored and academia developments and COTS products to provide users with an effective development environment. The following sections detail the software tools available for the EHPSCS Multicomputer Testbed.

To facilitate user support, Sanders developed a two-day Software Training Workshop on the use of the EHPSCS software tools and an application development process on the testbed. The workshop also includes an overview of the testbed hardware and software technologies employed with hands-on tutorials on MPI, Nupshot/Paragraph, DSP Libraries, and a sample application development and compilation on the testbed.

#### 4.2.1 Multiprocessor Debugger

To provide effective debugging capability for multicomputer application development on the EHPSCS architecture, the Dolphin Interconnect Systems TotalView debugger was adopted. TotalView is a source-level, multiprocessor debugger commercial product that is currently in use on several workstation and embedded platforms. It provides typical source-level debugger support, but across multiple processors. Key multiprocessor debug features include setting

breakpoints in multiprogram code, single stepping through the code, viewing the code at either the source level or assembly level, viewing the values of variables and memory locations, and setting the values of variables or memory locations. TotalView allows the user to simultaneously view and control multiple processes. TotalView is designed to be portable to different platforms by separating the target specific functionality from the core debugger functionality. Its product structure enables cost-effective adaptation by the EHPSCS project.

The EHPSCS version of TotalView runs on a Sun workstation and communicates to the SHARC target processors via Myrinet. The process of porting Totalview to the EHPSCS target involves a detailed model of the SHARC instruction set and a disassembler to convert the SHARC opcodes to assembly language instructions. Specific challenges involved the support of the SHARC's delayed branch instructions, which required proper identification and handling. Models of the SHARC's registers and runtime stack were also required. A set of routines to read the ADI Common Object File Format (COFF) executables also had to be developed to properly extract debugging information from the programs being debugged.

The EHPSCS communication message format was extended to provide a low-level communications protocol that allows TotalView to communicate with the SHARCs and to control SHARC execution for debug operations. The SHARC software library support was extended to include functions such as setting breakpoints, handling a breakpoint when it is encountered during runtime and reading/writing SHARC registers. These new SHARC library functions can be linked in with a SHARC executable to enable TotalView operations at runtime.

TotalView provides the user with the option of acting as a Sun host for an EHPSCS runtime session. Command line switches allow the user to specify whether TotalView should load the Sun SBUS card MCP code and/or the APU MCP and SHARC executables. The user must provide TotalView with three system configuration files in order for it to run correctly. These files specify a map of Myrinet network and a mapping of the executable processes to the processors they should run on. Two of these files, hostdb and routedb, are specified via a system environment variable and the other is specified on TotalView's command line.

TotalView was used extensively in the development and integration of the software for the Unmanned Underwater Vehicle (UUV) technology insertion program. This program consisted of a Sun host application and 52 APU processing nodes cooperating together to reduce data from a high-resolution sonar array. TotalView was found to be very effective in isolating and identifying bugs in the APU software, particularly during the software integration process.

For a detailed description of the TotalView debugger and user instructions, see the TotalView multiprocessor debugger User's Guide. The TotalView HPSC Release Notes are included as an attachment to this document.

### 4.2.2 Profiling Tools

The Sanders' MPI implementation contains various profiling libraries that can be used when either debugging or tuning the performance of an application. These libraries write profiling logs that can be read by the Nupshot profiling tools, developed by Argonne National Lab. With

the Transpicl tool developed by Lincoln Labs, these log scans be converted for use by the Paragraph profiling tool.

The MPI implementation has a predefined set of events that gets logged for profiling. Additionally, the developer may make use of the profiling library to define and log their own events. The profiling tools provide an excellent means of determining correct connectivity and load-balancing issues in the target application. Developers use these tools to increase the performance of their application. The tools can significantly improve productivity with an effective display of dynamic system behavior of a complex application. Events can be set and recorded, enabling users to accurately time each portion of their algorithm to analyze performance bottlenecks and resource utilization/efficiency.

#### 4.2.2.1 Nupshot

The Nupshot program illustrates the performance history of all processors in the network. It is an updated version of Upshot which is a public domain X Windows-based parallel program visualization tool developed by Argonne National Laboratory. The data is logged in the Alog format as described in the Upshot documentation. As shown in the display in Figure 37, the processors are identified top to bottom along the vertical axis, starting at process 0 and ending at process N-1. Time is displayed left to right along the horizontal axis. Colored boxes indicate the various states of a process.

Nupshot correlates the communications transfers so developers can visually see when processors are intercommunicating and how long it takes for each communication. Each discrete event is color-coded. This makes it easier for the developer to concentrate on a particular type of event without re-compiling and re-running the application.

**Nupshot**
- Public domain X windows based parallel program visualization tool
- Developed by Argonne National Laboratory
- Nupshot, a newer version, is distributed with MPICH
- Performs post-mortem analysis on log files generated by parallel MPICH programs
- Log files are in the Alog format as described in the Upshot documentation.

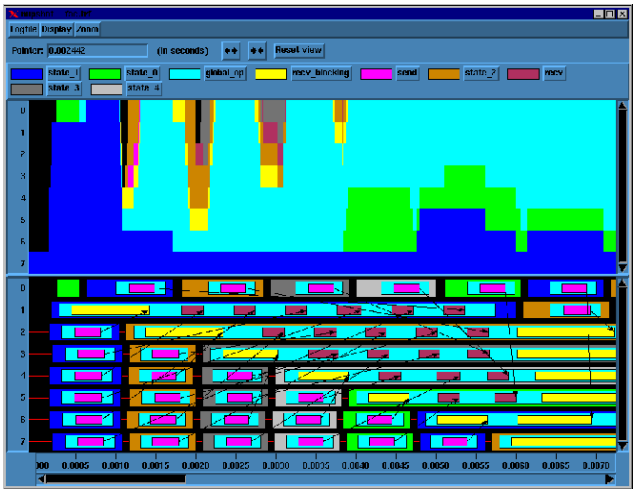


**Figure 37. Nupshot Profiler**

#### 4.2.2.2 ParaGraph

The ParaGraph analysis tool displays similar information as Nupshot, but uses a more dynamic approach. The Profiling data may be played back through time, whereas Nupshot provides a static timeline of when events occurred. Different displays can be used to view the processor topology and network operations in different ways. ParaGraph provides a visual means of detecting instantaneous unbalanced loads in a parallelized algorithm, and allows the developer to determine undesirable effects in the processing timeline. A view of ParaGraph is shown in Figure 38. **ParaGraph Profiler**

Lincoln Labs has augmented ParaGraph to include an EHPSCS chassis-specific network topology. Additionally, Lincoln Labs has created the Transpicl tools to convert Alog files into PICL format for animation in the ParaGraph tool.

**ParaGraph**

- An animated visualization tool to analyze the behavior of parallel programs
- PICL and the data file format were developed at Oak Ridge National Laboratory
- PICL execution trace data can be replayed through ParaGraph providing visual animation
- Inputs event data accumulated by the Portable Instrumented Communication Library (PICL)
- Work is currently underway at MIT Lincoln Laboratory to build extensions to ParaGraph for visualizing the Advanced Common Processor (ACP) execution as well as translation tools to convert Alog files from MPICH programs into PICL format for animation ParaGraph tool.

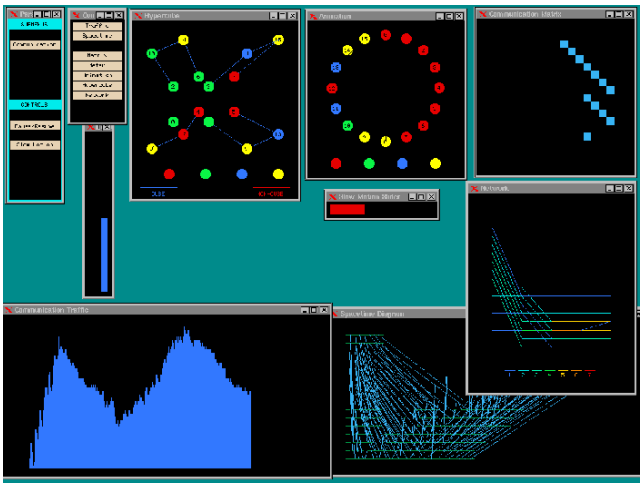


**Figure 38. ParaGraph Profiler**

### 4.2.3 Architectural Simulation and Analysis

In conceptualizing and designing a system, it is important to have various simulation capabilities to help in making system design decisions; this is particularly true for high performance multicomputer systems. A performance modeling capability allows the development team to assess the performance and impact of various architectural design decisions. As part of the EHPSCS effort, Sanders has developed performance modeling, simulation, and analysis capabilities using the Ptolemy and RAMP tools.

#### 4.2.3.1 Ptolemy

Ptolemy is a software environment developed at the University of California, Berkeley that supports heterogeneous system simulation and design using several different models of computation, each implemented in a separate domain. The EHPSCS performance modeling capability uses the Ptolemy's Discrete Event (DE) domain as its engine for performance simulation. This capability has been demonstrated and a paper on the capability was presented at the 1997 IASTED International Conference on Modeling and Simulation. The paper by Eric K. Pauer is entitled "High Performance Scalable Computing Performance Modeling Using Ptolemy", and is attached to this document.

The DE domain is a discrete-event simulator that uses a model of computation in which tokens with time stamps, called particles, which represent events, are passed among the simulation building blocks, called stars. These are shown in Figure 39. **Myrinet Performance Modeling Stars** Ptolemy has been developed in C++ using an object-oriented software architecture to facilitate modularity and extensibility. All of the source code in Ptolemy is freely available via the World Wide Web, which facilitates adding extensions to the tool. Extensions to the DE domain, in the form of new stars and particles, were created for the EHPSCS architecture and Myrinet protocol. This effort leveraged off similar performance modeling work started here at Sanders under the RASSP program, in addition to the work already done at the University of California under the Ptolemy project.

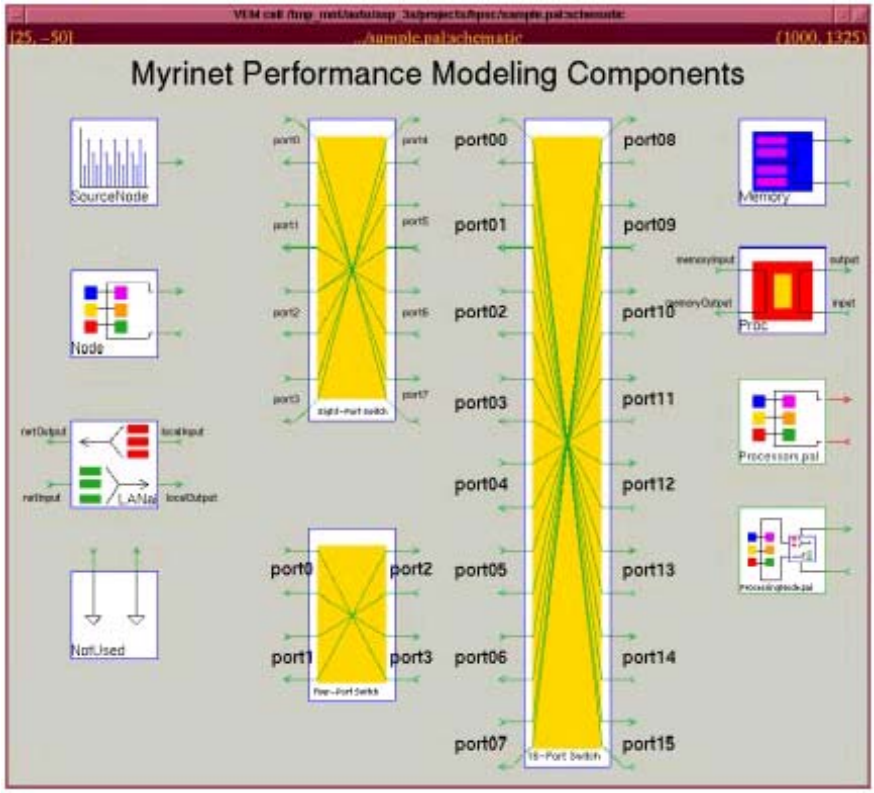



**Figure 39. Myrinet Performance Modeling Stars**

The key components in the EHPSCS architecture include data sources (SourceNode), LANai interfaces (LANai), processing nodes (Node), and Myrinet switches (4/8/16-port Switches). The stars are behavioral virtual prototypes of the components, as they implement behavioral models at the appropriate level of abstraction. Each type of star has a group of settable state parameters, which allow the behavior of the model to be adjusted or fine-tuned as appropriate. In addition to the new stars, several new particles were developed to model the data blocks passed between the Nodes and LANais and the various Myrinet data and control packets passed among the LANais and switches. These new stars and particles, combined with the built-in stars available with Ptolemy, allow performance simulations of large, complex scalable systems to be modeled for analysis.

An additional strength of Ptolemy is its support for hierarchical modeling. Groups of stars and their interconnections can be captured into a single entity called a galaxy. A galaxy facilitates both reuse and simplifies the structure of complex system models. This hierarchy makes the task of creating, managing, and simulating large architectures much easier. The previously referenced IASTED paper provides several examples of this hierarchy.

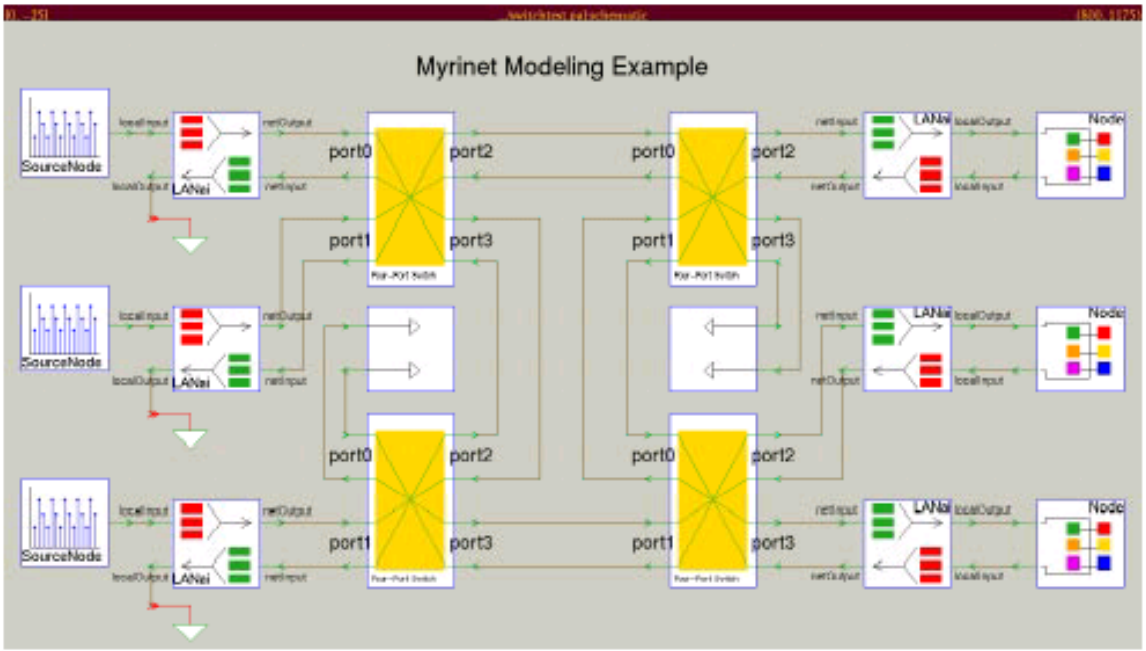



**Figure 40. Simple Myrinet Modeling Example**

Using the Myrinet models, the various components are placed and connected to specify a system architecture. One example is shown in Figure 40. **Simple Myrinet Modeling Example** All systems must include at least one SourceNode star, representing the source of data into the system. Each SourceNode is paired with a LANai star. As a high-level model of a data source, the Source star generates data blocks of a specified size at a periodic rate. The system also typically includes many Node stars, each also paired with a LANai star. The Node star models a processing node at a high-level abstraction, treating the processing taking place on the Node as a single measurable task. In both cases, the LANai acts as an interface between the Node or

SourceNode to the Myrinet network. Lastly, among the LANai stars paired with the Node and SourceNode stars, there is a network of Myrinet Switch stars, representing the network topology of the system. There are currently models for four-,eight-, and sixteen-port switches, and switches with a different number of ports can be easily supported.

In order to more easily view and interpret the results of the simulation, a Gantt tool was developed. The Gantt tool displays the activity on each resource in rows over time (time is along the x-axis), shown in Figure 41. **Gantt Tool Display of Simple Myrinet Modeling Example** There is a row for each SourceNode generation of data, each LANai transmit activity, each LANai receive activity, the transmit activity for each port in every switch, the transmit queue for each port in each switch, and the processing on the node. Thus, most of the stars need several rows to display their behavior and performance. Rows are not displayed when there is no activity, but the displaying of rows may be disabled. The various activities have been color-coded to facilitate viewing. Yellow denotes a start up latency, blue indicates normal transmission or reception of data, and green indicates processing of data by the node. Problems are shown in orange and red. Orange indicates that one or more blocks have currently originated in the switch port and have caused queuing of requests; red is used where switch ports or LANais are idle due to blocks that occurred somewhere in the current route path. There are also labels containing two integers on most activities. The first number indicates the data packet's relative position within the transmit DST in the LANai where it was transmitted. This first number of the packet label is different when it is displayed by the receiving LANai, in which case it indicates the relative index of the data packet in the receive DST. The second integer is a unique global identification number assigned to the packet. These numbers are assigned sequentially as packets and are created in a given simulation; no two data packets will have the same number. This identification number facilitates the tracing of a given packet through the Gantt display from the transmitting LANai, through the switches, to the receiving LANai.


**Figure 41. Gantt Tool Display of Simple Myrinet Modeling Example**

This capability enhances the ability of the designer to explore many options in order to find the EHPSCS architecture that best satisfies their system requirements. For more detailed information on the Ptolemy tool, refer to http://ptolemy.eecs.berkeley.edu.

#### 4.2.3.2 RAMP (Real-time Algorithm Mapper and Performance analyzer)

The Real-time Algorithm Mapper and Performance Analyzer (RAMP) tool was developed by General Electric Corporate Research and Development, under the funding of Lockheed Martin. RAMP is a graphical tool for designing multi-processor based systems, which helps in evaluating the suitability of architectures for implementing algorithms. RAMP provides a methodology and tools for rapidly developing real-time systems (such as signal/image processors, and avionics) from reusable hardware and software modules. RAMP supports performance optimization and software reuse. It has a graphical interface consisting of an algorithm window and an architecture window. Much like the RASSP architectural trade capability, the user maps the algorithmic blocks onto the architecture. For more information, refer to "An Architectural Trade Capability Using the Ptolemy Kernel", attached to this document. RAMP provides an automatic routing capability, using a shortest route assignment, for the initial flow of data on the architecture. It provides the capability to import algorithm topologies exported from Alta's Signal Processing Workstation (SPW). One disadvantage of RAMP is that it does not easily allow the addition of models or modeling at higher or lower levels of abstraction. The cost functions are also a little constrained as they are not proportioned to the amount of data being processed. A built-in discrete event simulator is integrated into the tool and provides the simulation capability for the mapped architecture.

During the initial architecture design phase of the project, RAMP was used to analyze the network topology for the testbed design. RAMP was also explored as an architectural 'editor' to provide users with a graphical user interface (GUI) for defining a network topology. Although RAMP cannot model the Myrinet protocol at the desired level of abstraction, it can export this data to serve as a GUI front end tool for performance simulation under Ptolemy. For more information on the RAMP tool, refer to http://www.sanders.lmco.com/at/hpcot/ramp/index.html

### 4.3 ReConfigurable Transport Engine (RCTE)

In the second half of the EHPSCS program, the baseline two-level multicomputer architecture was extended to specifically address multiple next-generation application challenges. Key challenges included technology-neutral support and hard real-time performance for wide range support of high performance applications. The key objective of the RCTE design is to implement the extended architecture features in a prototype to functionally test and validate the hardware prototype, and to demonstrate and characterize performance that can be compared to similar measurements on the LANai-based APU.

#### 4.3.1 RCTE Concept Overview

Key network controller enhancements offered by the RCTE over the APU microarchitecture include zero-copy network communications overhead, reconfigurable host and network interface support, and reconfigurable hardware for real-time data format conversions. These offer many advantages over the LANai based network interface such as a 10x improvement in

communication latency, technology neutral support of network interfaces or processing resources, COTS network processor support, and reconfigurability for application optimization. The enhancements were derived from lessons learned in the performance, operation, and usage of the APU-based EHPSCS testbed.

The RCTE concept is shown in Figure 42. The RCTE function decouples the network protocol interface from the resource, consistent with the EHPSCS architecture approach. That function is reconfigurable such that different networks and resource technologies can be supported as well as application specific functionality such as protocol acceleration, data format conversion, or performance monitoring. This promotes rapid technology insertion by relying on firmware modifications at most for new resources or functionality.

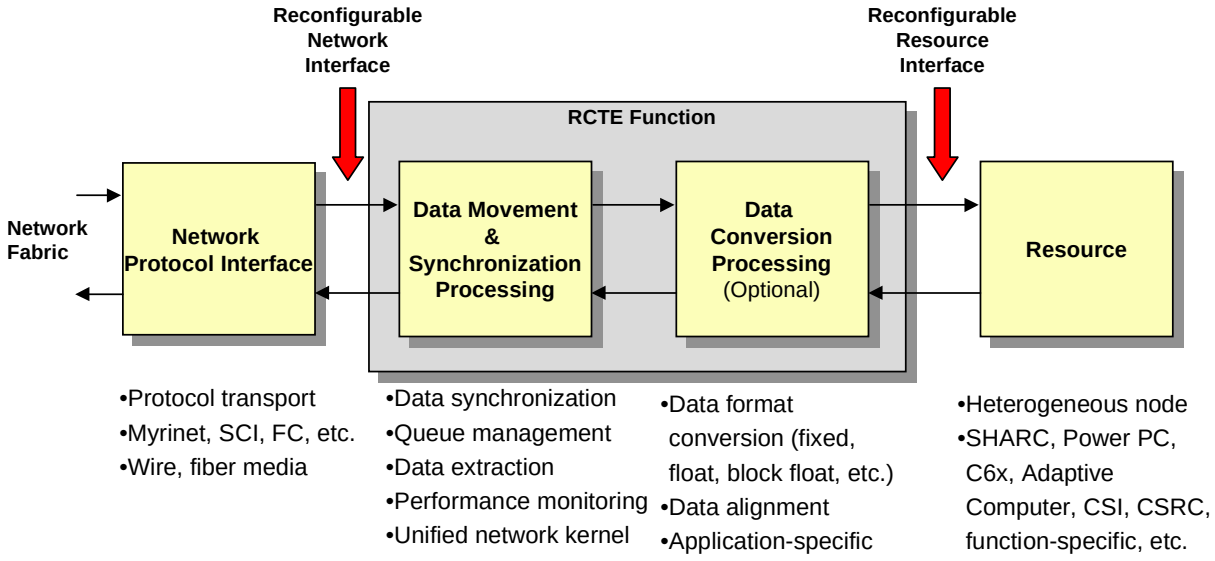



**Figure 42. ReConfigurable Transport Engine Concept**

All network data passes directly through the RCTE with zero copies. Messages need not be copied to network controller memory before transmission onto the network nor into network controller memory upon reception.

### 4.3.2 RCTE Microarchitecture

A block diagram of the RCTE network interface is shown in Figure 43. This implementation incorporates a Myrinet network protocol interface, a Motorola MPC860 PowerPC microcontroller, and an FPGA-based Data Synchronization/Direct Memory Access (DSE/DMA) Engine.

Myrinet was chosen as the network protocol to retain compatibility with the existing EHPSCS testbed. It is implemented with the FI chip from Myricom. The FI has bidirectional, full duplex support for the Myrinet protocol and a simple FIFO interface.

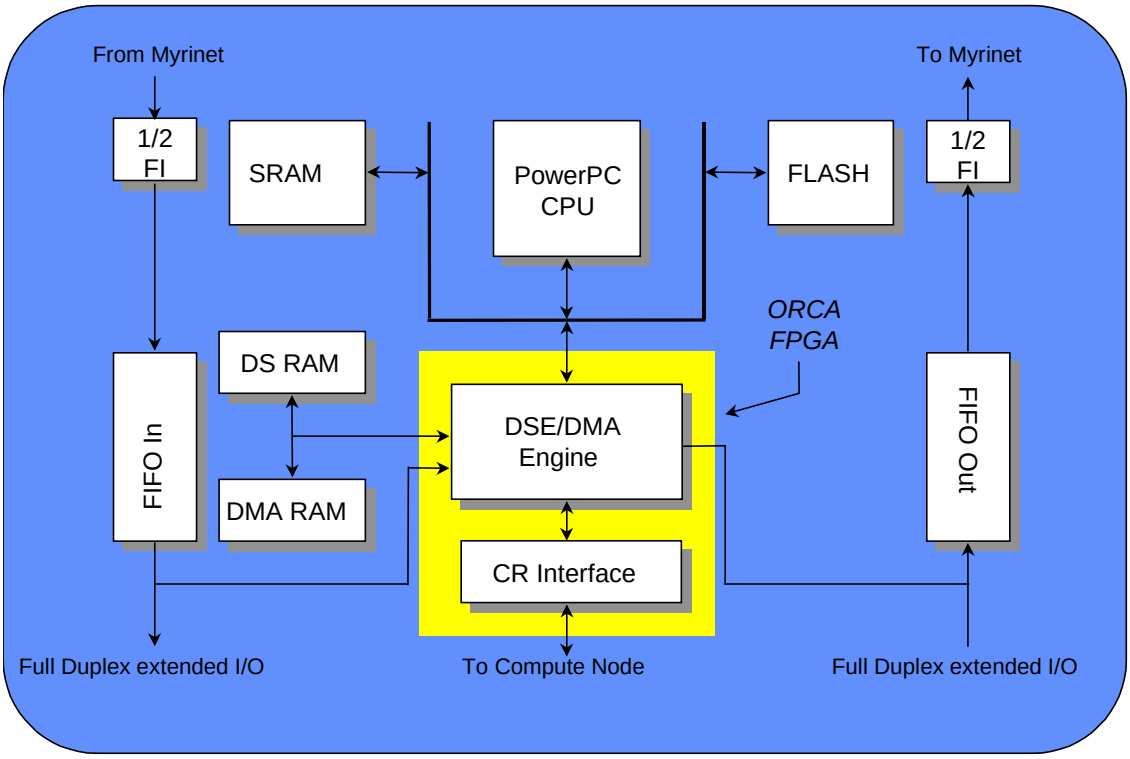



**Figure 43. ReConfigurable Transport Engine Network Controller**

The DSE/DMA engine was developed entirely in modular VHDL such that support of a different resource interface or application specific function requires replacement of that module's VHDL. This engine was developed with support for a SHARC-based resource as the Compute Node interface and a hardware acceleration scheme for the Data Synchronization Queue protocol discussed in Section 4.1.5. The DSE/DMA engine is notified when a data set is received or ready to be transmitted and, through the use of lookup tables, DMAs the data to its destination on the resource bus or out to the network. This protocol hardware acceleration, coupled with the zero-copy architecture, results in a predicted network overhead latency reduction of a factor of 10 over the LANai-based APU. The actual measured results are discussed in Section 4.3.5 The movement of messages that are not identified as data sets is the responsibility of the network controller.

The MPC860 microcontroller performs general housekeeping, lookup table management, and the processing of any non-data set messages. A wealth of COTS support exists for the MPC860 in the areas of tools, RTOSes, technical support, etc. Sanders reviewed several choices for emulators and RTOSes and decided on the VisionICE Development System from Embedded Support Tools Corporation and the pSOS operating system from Integrated Systems, Inc.

Two of these network controllers were implemented on the RCTE motherboard, connected via a Myrinet crossbar switch as shown in Figure 44. The motherboard is fully compatible with the EHPSCS testbed. To take advantage of the reconfigurability on the resource side, the resource is implemented on a daughter card. The PCI Mezzanine Card (PMC) standard was adopted for the mechanical specification of the daughter card sites to take advantage of COTS resources

available in PMC format. A SHARC-based daughter card was developed for the prototype due to Sanders experience with the processor and the tools and software already in place from the APU development. The daughter card scheme provides a platform to mix and match resources with a replacement of the resource interface VHDL module in the DS/DMA engine, greatly reducing development time and therefore technology insertion time.

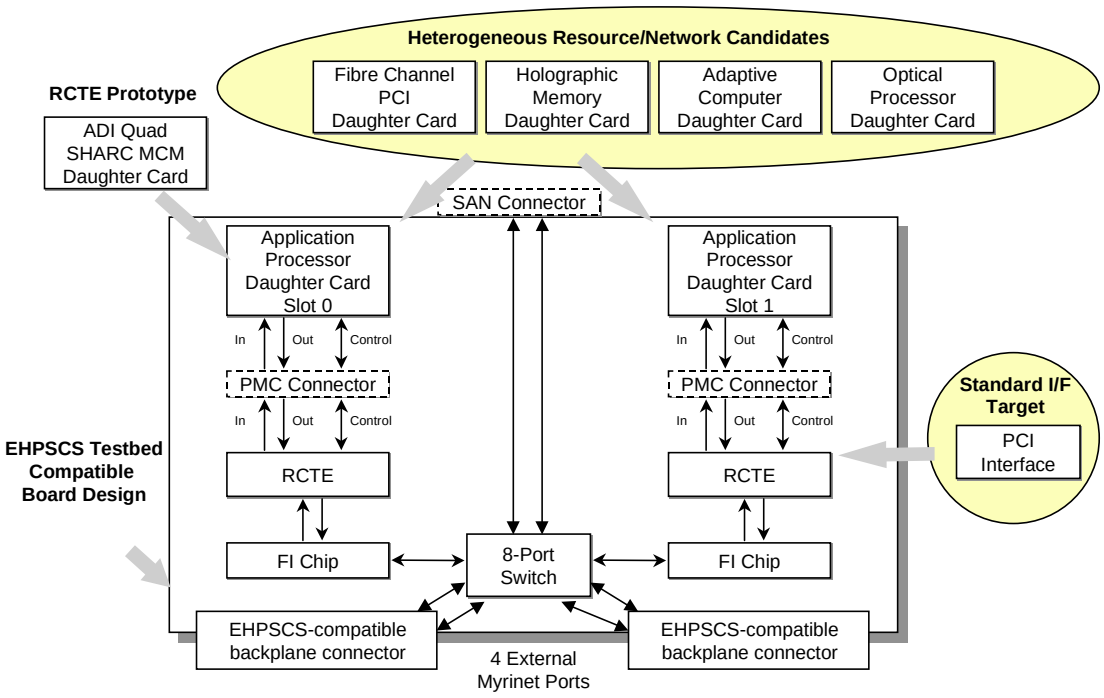



**Figure 44. RCTE Motherboard block diagram**

Conceptually, the RCTE motherboard, populated with Quad-SHARC-based daughter cards, is the same functionality as the APU board, but with higher network performance and greater flexibility. Figure 45 below compares the two boards, as well as a flat multicomputer, to show the improvement in communication latency and resource efficiency from a flat multicomputer to a LANai-based, two-level multicomputer to the RCTE-based, two-level multicomputer.

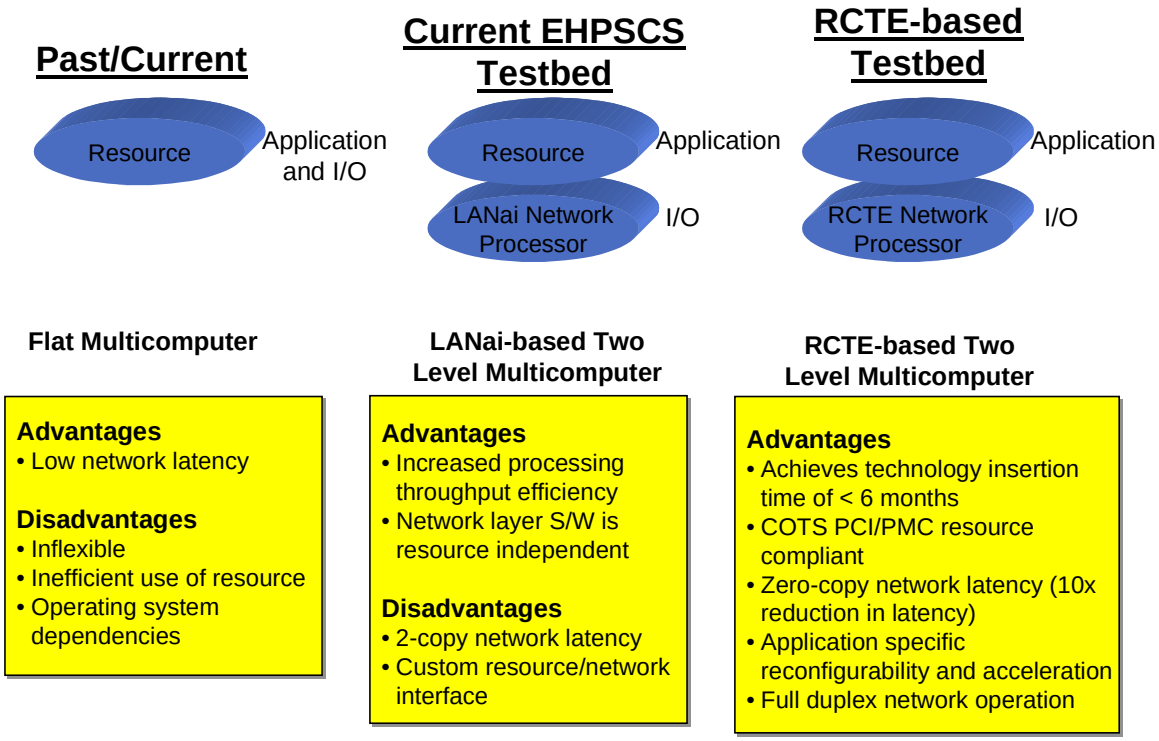



**Figure 45. Multicomputer Interface Comparison**

For more detailed information on the RCTE and its operation, refer to the ReConfigurable Transport Engine Technical Specification attachment at the end of this document.

### 4.3.3 RCTE Hardware

The RCTE module was a prototype developed for the performance characterization and validation of the key architectural enhancements. Two RCTE motherboards were built as well as two SHARC-based resource daughter cards. The verification processes for these two assemblies was performed in parallel. Photos of the two assemblies are shown in Figure 46.

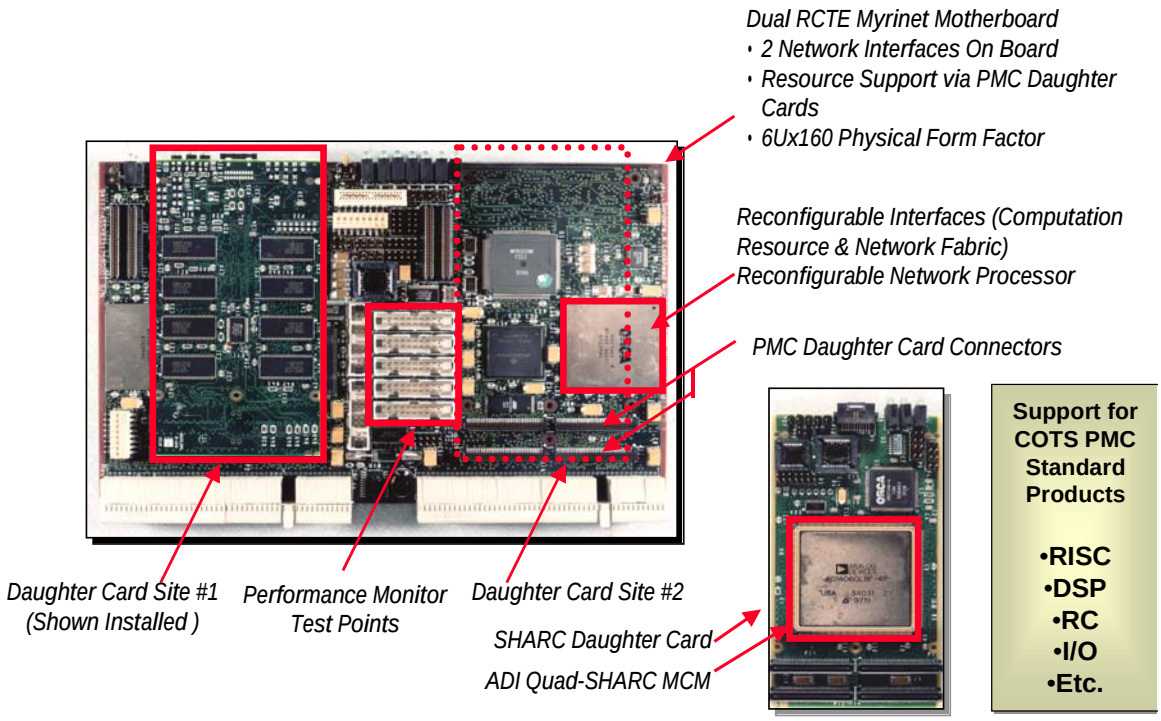



**Figure 46. RCTE Motherboard and SHARC Daughter Card Assemblies**

The development of the SHARC daughter card was largely based on the APU design to serve as a low-risk, consistent resource function to support the RCTE endeavor. The existing debug monitor software was adapted with minor modification to accommodate the link port connectivity specific to the ADI SHARC MCM. The hardware design consists of an enhanced DRAM controller function for the support of synchronous DRAM, deviating from the APU design. The hardware verification of both daughter cards was completed in November 1997. As can be seen in Figure 47, the board consists of a commercially available Quad SHARC MCM, 64 MB of SDRAM, and 512 KB of nonvolatile FLASH memory.

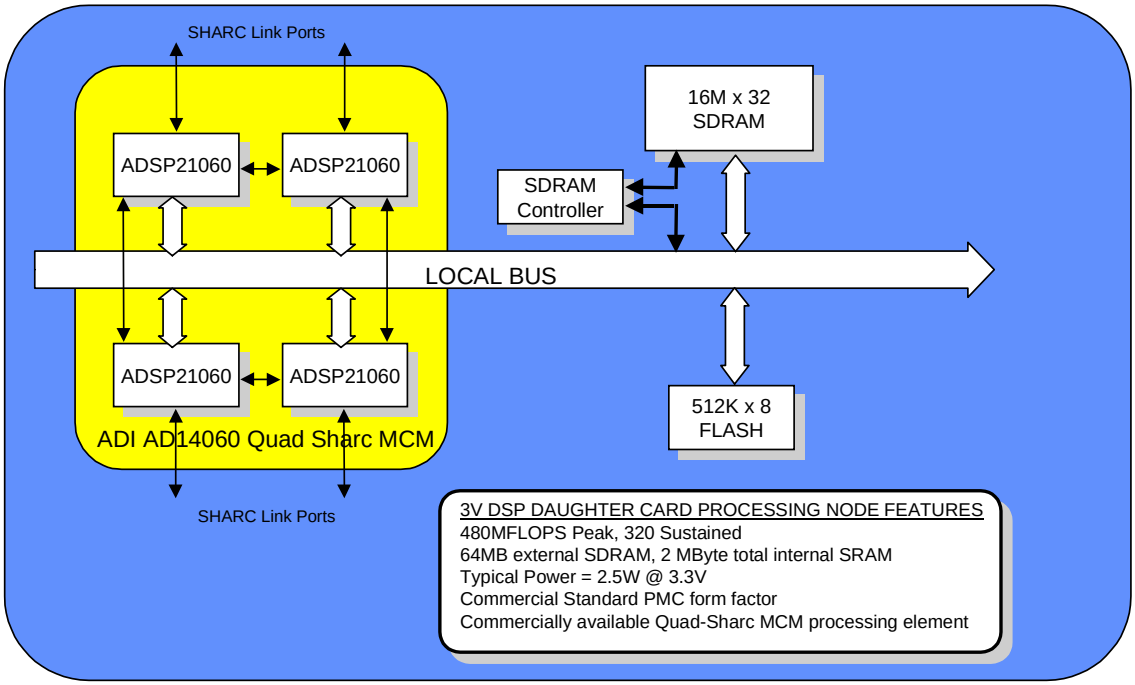



**Figure 47. Quad SHARC Daughter Card Functional Block Diagram**

As shown in Figure 43, the network data path of RCTE Network Controller is FIFO based, which reduces communication latency by eliminating copies of network packets into SRAM. The FIFOs chosen for the Network Controller are 8K words deep to minimize any interruptions in data transfer on both the resource and network sides.

Both the CPU and the DSE/DMA Engine have access to 128K words of synchronous SRAM. The DS RAM and DMA RAM are used to store parameter tables for incoming and outgoing data sets. The CPU also has access to 512K bytes of FLASH memory to store boot code as well as the FPGA configuration. The CPU is responsible for FPGA configuration on powerup.

Special attention was paid to the separation of three clock domains within the Network Controller. Separate clocks support the network, the CPU, and the resource. The DSE/DMA Engine was designed to support a resource clock domain separate from the CPU clock domain to maximize flexibility and performance. The FIFOs decouple the network clock domain from the resource clock domain. The separation of clock domains maximizes the performance of each portion of the Network Controller by allowing them to run at their maximum capacity. The separation also allows flexibility in the choice of resources.

A series of emulator- and network-based tests were used to verify the hardware functionality of the RCTE Motherboard. The reconfigurability of the RCTE was emphasized during hardware verification. Different configurations were developed and used to exercise functions as well as pinpoint errors in network communication tests. The hardware functionality was ultimately verified in recognizing and processing data set packets and non-data set packets. A variety of data sets were constructed, varying in data slot size and data set size, to fully exercise the DSE.

This was the basis of the performance characterization of the DSQ hardware acceleration functionality. The results of the characterization are described in Section 4.3.5.1.

### 4.3.4 RCTE Software

The software developed for the RCTE under the EHPSCS program was designed to provide a functional demonstration of the key RCTE features. This demonstration employs a round-trip, data set transfer scenario between the Sun host and two RCTE SHARC nodes connected via Myrinet. It verifies the data set handling functionality of the RCTE, which is representative of the typical operation of this hardware.

For the round-trip demonstration, an ASCII configuration file must be created defining both the outgoing (i.e., producer) and the incoming (consumer) data sets. The Sun host accepts this ASCII configuration file and generates all the tables required for driving the RCTE. The Sun host EHPSCS application begins the demonstration by issuing EHPSCS resource messages to load the RCTE tables across the Myrinet network. The 860_DARC is the embedded program that runs in the RCTE's MPC860 embedded controller. It receives and processes the resource messages from the Sun EHPSCS application. Once the RCTE tables are loaded in the RCTE RAM, the Sun EHPSCS application starts the SHARC application by sending another resource message. The SHARC application produces the data sets and initiates the transfer of the first data sets. These outgoing data sets are transferred from the first RCTE node to the second RCTE node. The second RCTE node receives the (now incoming) data sets, processes them, and transfers them back to the first RCTE node. The second RCTE node receives the returning data sets, which completes the round-trip demonstration. These processes are shown in Figure 48. **Sun Host Utility Program Static Table Generation**, Figure 49. **RCTE RAM Table Load**, and Figure 50. **Data set Roundtrip**

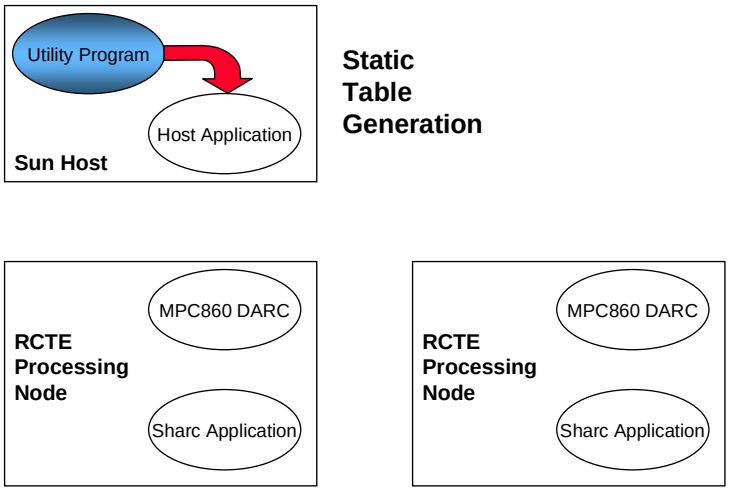



**Figure 48. Sun Host Utility Program Static Table Generation**

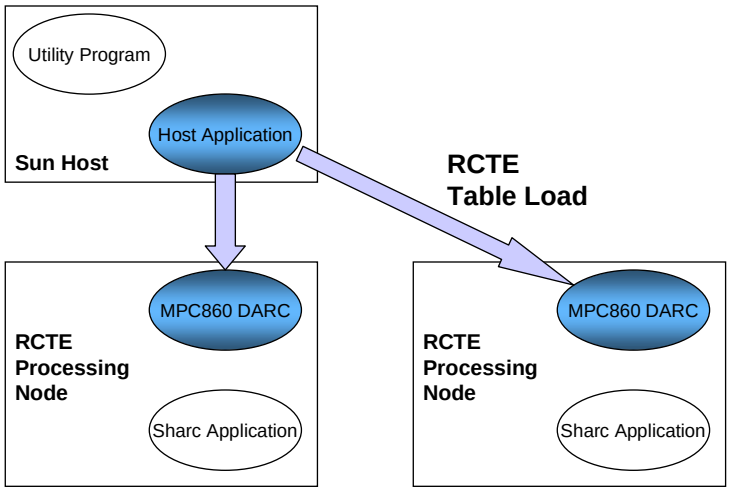



**Figure 49. RCTE RAM Table Load**

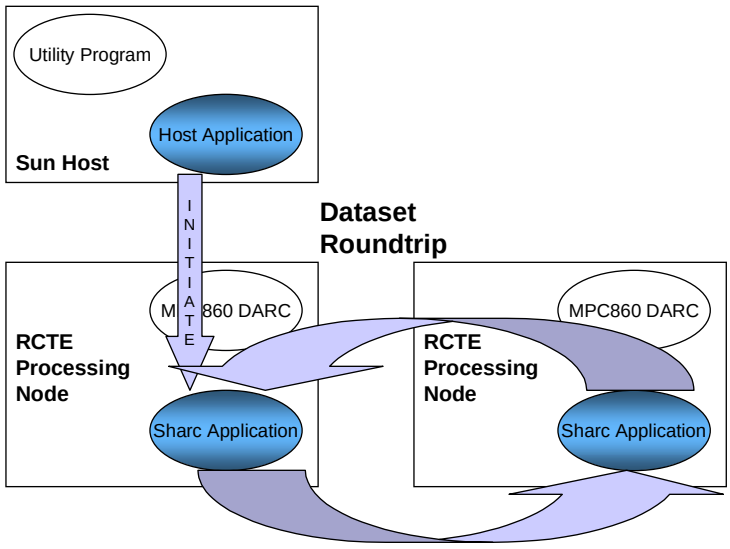



**Figure 50. Data set Roundtrip**

The five blocks of software designed for the RCTE are described as follows.

**Sun Host Utility program:**

This Sun SPARC host resident application is an off-line utility program that has been 100% developed and tested. It is a tool for obtaining user input and generating the required RCTE data structures for the demonstration scenario. The user provides input in the form of a text configuration file. The program accepts this text file as input and outputs the generated RCTE tables in individual binary (and Motorola S-Record formatted text) files.

**Sun Host EHPSCS Application:**

This Sun host resident application is linked with a set of modified EHPSCS SUN_RNI libraries and uses a subset of the SUN_RNI API to send packet messages to the RCTE via Myrinet. This application loads the RCTE tables and uses the following API functions:

SLM_init

SLM_RCTE_Write*

SLM_RCTE_Read*

SLM_RSRC_Send

SLM_RSRC_Recv

MMSG_Prog_write

MMSG_Prog_Start

These functions are new additions to the SUN_RNI API for the RCTE. The modified SUN_RNI libraries implement SLM_RSRC_SEND (much like SML_DEBUG_Write and SLM_DEBUG_Read) to perform the loading the RCTE Data tables from the Sun host in to RCTE RAM.

**860_DARC - an MPC860 QUICC Embedded Application:**

This application executes on the MPC860 and functions as a Myrinet Control Program (MCP). The embedded application and the RCTE HW together provide the Distributed Architecture Resource Controller (DARC) functionality in this microcontroller. This 860_DARC:

- Holds the compute resources (SHARC DSPs) in reset state.
- Initializes the RCTE hardware registers.
- Receives resource packets via Myrinet and load the RCTE data tables into RAM.
- Releases the compute resources (SHARC DSPs) from reset and load initial SHARC boot program.

- Receives resource packets via Myrinet and initiate the SHARC DMA to pack and load the 48-bit SHARC application into SHARC RAM.
- Handles MPC860 interrupts to handle errors and service both SHARC messages in the 860 FIFO and resource packets from the Myrinet network in the Network input FIFO.
- Processes messages received from the Myrinet network and SHARC resources.

**SHARC Application**

This application running on the SHARCs processes incoming and creates outgoing data sets via handshaking with the RCTE during the demonstration. The SHARCs then receive an incoming data set notification from the RCTE when the round-trip transfer has completed.

**SHARC RCTE Support**

This component was developed to support the higher level SHARC support library by performing the RCTE specific functions. These functions include handling SHARC Inter-Processor Communication (IPC) mailbox interrupts, reading the (IPC) mailbox, and sending RCTE Data Synchronization Engine (DSE) notifications.

All of the software design and implementation is complete. Due to overruns in the hardware schedule and to the hardware intensive nature of the RCTE, the software remains untested on the RCTE board. Both the RCTE hardware and performance validation were sufficiently demonstrated for data sets without the use of the described software.

### 4.3.5 RCTE Performance

From a performance standpoint, the goal of the RCTE was to improve upon the latency and data buffering overheads of the LANai-based message passing network architecture and realize additional performance and resource efficiency for many hard real-time applications. The performance goal was an order of magnitude reduction in Data Synchronization Queue communication latency over the existing LANai-based APU. In addition, the RCTE seeks to demonstrate an improvement in power performance over the APU network controller function. The following characterization data show Sanders success in meeting these goals.

#### 4.3.5.1 RCTE Latency Performance

Characterization of the latency and DSQ acceleration performance consisted of loading predefined lookup tables to the Data Synchronization Engine to support a round trip transfer of a data set. The round trip between Node A and Node B on a single motherboard was bounded by timestamped messages to mimic and measure a typical send/receive message passing function. Figure 51 shows the complete message passing used to characterize the latency performance. These time-stamped packets were then passed on to the Sun host and read for analysis.

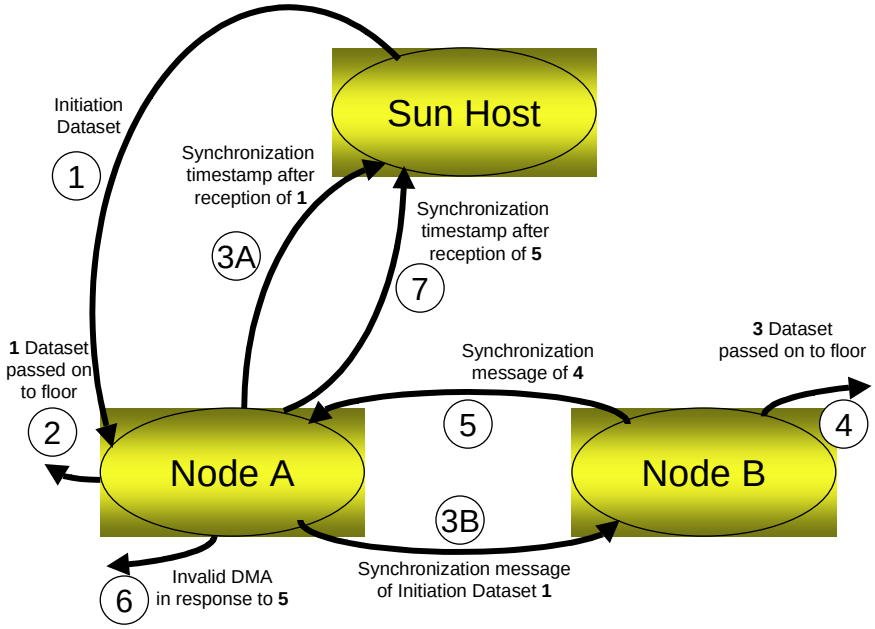



**Figure 51. RCTE Message Passing for Characterization**

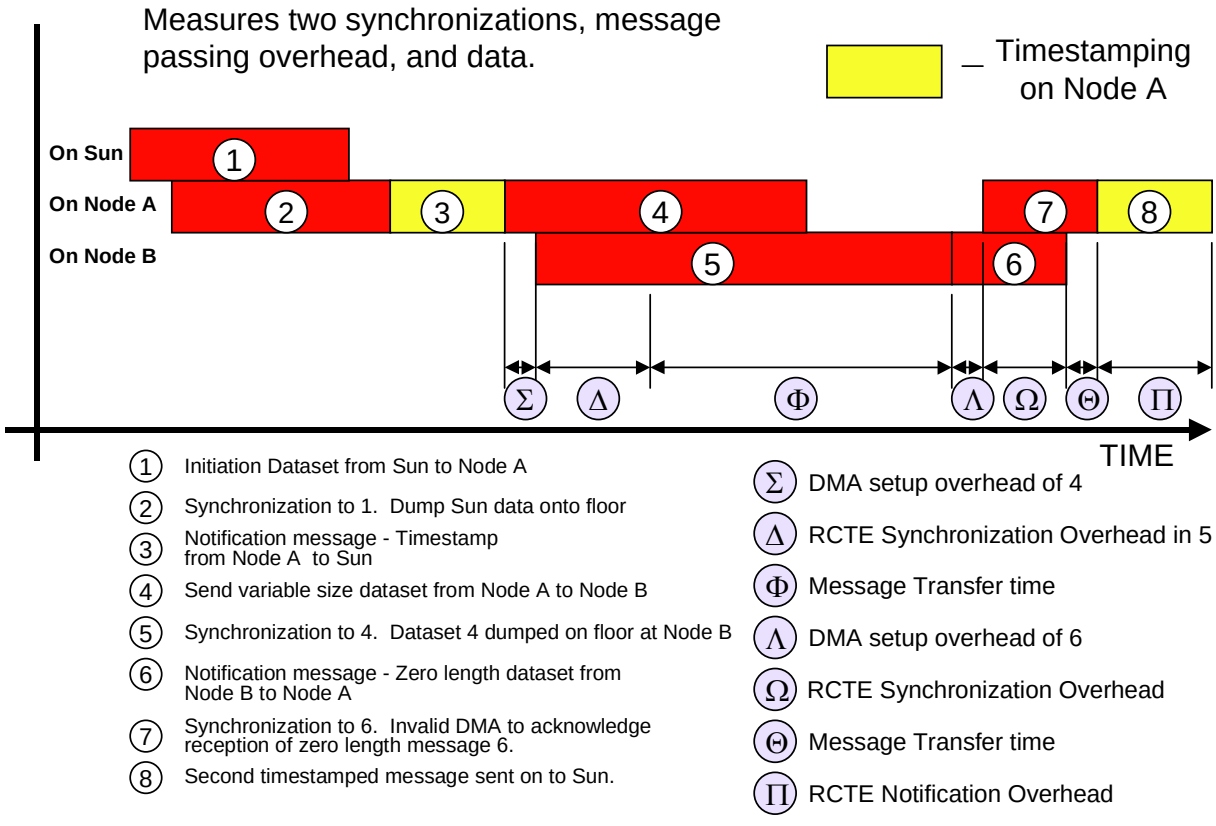



**Figure 52. RCTE Latency Measurement Timeline**

The latency measurements were made using one node (Node A) as the timing reference. Figure 52 shows these message transfers on a timeline to visualize the operations being timed. The communication latency was measured as the time to send a message (4) from Node A, receive and synchronize (5) at Node B, notify with a message back to Node A, and to synchronize (7) and notify (8) on the received message back from Node B at Node A. Messages (3) and (8) contain only a timestamp for a body. Messages (4) and (5) were used to vary data slot and data set sizes. Message (6) contained no body and only consist of header information. This configuration easily provides message passing overhead latency measurements while most closely modeling a send/receive pair. The results of these measurements are shown in Table 2. Note that in the following paragraphs, Total Latency is defined as the time to transfer a message from resource A to resource B, including the actual data transfer time. Overhead Latency is defined as the time to transfer a message from resource A to resource B, excluding the actual data transfer time. Overhead latency + (message size * 25ns) = Total Latency.

| **RCTE Total Latency Measurements** | | | | | | |
|---|---|---|---|---|---|---|
| **Slot Size (Bytes)** | **Slot Count** | | | | | |
| | **1** | **4** | **8** | **16** | **32** | **64** |
| **4** | 6.68 | 11.68 | 21.05 | 39.85 | 77.45 | 152.65 |
| **16** | 6.75 | 11.95 | 21.68 | 41.05 | 79.88 | 157.45 |
| **32** | 6.85 | 12.38 | 22.45 | 42.65 | 83.08 | 163.88 |
| **64** | 7.05 | 13.15 | 24.05 | 45.88 | 89.45 | 176.68 |
| **128** | 7.45 | 14.75 | 27.25 | 52.25 | 102.25 | 202.25 |
| **256** | 8.25 | 17.95 | 33.65 | 65.05 | 127.85 | 253.45 |
| **512** | 9.85 | 24.35 | 46.45 | 90.65 | 179.05 | 355.85 |
| **1024** | 13.05 | 37.15 | 72.08 | 141.88 | 281.45 | 560.65 |
| **2048** | 19.45 | 62.75 | 123.28 | 244.25 | 486.25 | 970.25 |
| **4096** | 32.28 | 113.95 | 225.65 | 449.05 | 895.85 | 1789.45 |
| **8192** | 57.85 | 216.38 | 430.45 | 858.65 | 1715.05 | 3427.85 |
| **16384** | 109.05 | 421.15 | 840.05 | 1677.85 | 3353.45 | 6704.65 |
| **32768** | 211.45 | 830.78 | 1659.25 | 3316.25 | 6630.25 | 13258.25 |

**Table 2. RCTE Total Latency Measurements**

| APU Total Latency Measurements | | | | | | |
|---|---|---|---|---|---|---|
| **Slot Size (Bytes)** | **Slot Count** | | | | | |
| | **1** | **4** | **8** | **16** | **32** | **64** |
| **0** | N/A | N/A | N/A | N/A | N/A | N/A |
| **4** | 59.68 | 87.79 | 123.92 | 196.42 | 340.94 | 630.51 |
| **16** | 60.77 | 87.76 | 123.87 | 196.38 | 342.16 | 632.78 |
| **32** | 61.10 | 88.02 | 124.39 | 198.10 | 344.40 | 637.29 |
| **64** | 60.93 | 88.39 | 125.92 | 200.06 | 348.20 | 645.59 |
| **128** | 61.49 | 89.92 | 127.82 | 203.80 | 355.20 | 662.22 |
| **256** | 62.21 | 92.51 | 132.31 | 211.88 | 372.00 | 690.71 |
| **512** | 63.65 | 96.79 | 140.21 | 227.79 | 410.77 | 749.79 |
| **1024** | 67.82 | 109.08 | 166.74 | 280.67 | 508.55 | 967.93 |
| **2048** | 82.44 | 144.21 | 226.33 | 390.80 | 721.13 | 1380.06 |
| **4096** | 118.05 | 219.65 | 352.71 | 620.28 | 1153.40 | 2222.01 |
| **8192** | 195.49 | 371.42 | 608.17 | 1079.52 | 2024.64 | N/A |
| **16384** | 347.96 | 680.23 | 1121.05 | 2004.27 | N/A | N/A |
| **32768** | 658.40 | 1298.09 | 2152.15 | N/A | N/A | N/A |

**Table 3. APU Total Latency Measurements**

The most critical piece of data measured on the RCTE was the message passing latency for a 4-byte, single slot data set (pure overhead latency plus a single cycle – 25ns – of data transfer time on the network). This measurement on the RCTE was 6.68 us, compared to 59.68 us on the LANai-based APU – an improvement of 9X, just shy of the predicted results. It is important to note that the overhead latency of the RCTE is fixed. The overhead latency of the LANai is dependent on the transfer size of the data. This is due to the two-copy architecture of the LANai. Based on that, it can be shown that the RCTE overhead latency improvement over the LANai actually increases as data slot size increases.

As data slot sizes increase and the number of data slots per data set increase, the latency ratio declines as the LANai's two-copy architecture is masked by pipelining effects and latency times become dominated by the data transfer time. Even at the large data slot size in multi-data slot data sets, the RCTE still shows performance increases over the LANai-based APU. These trends are shown in the chart in Figure 53.

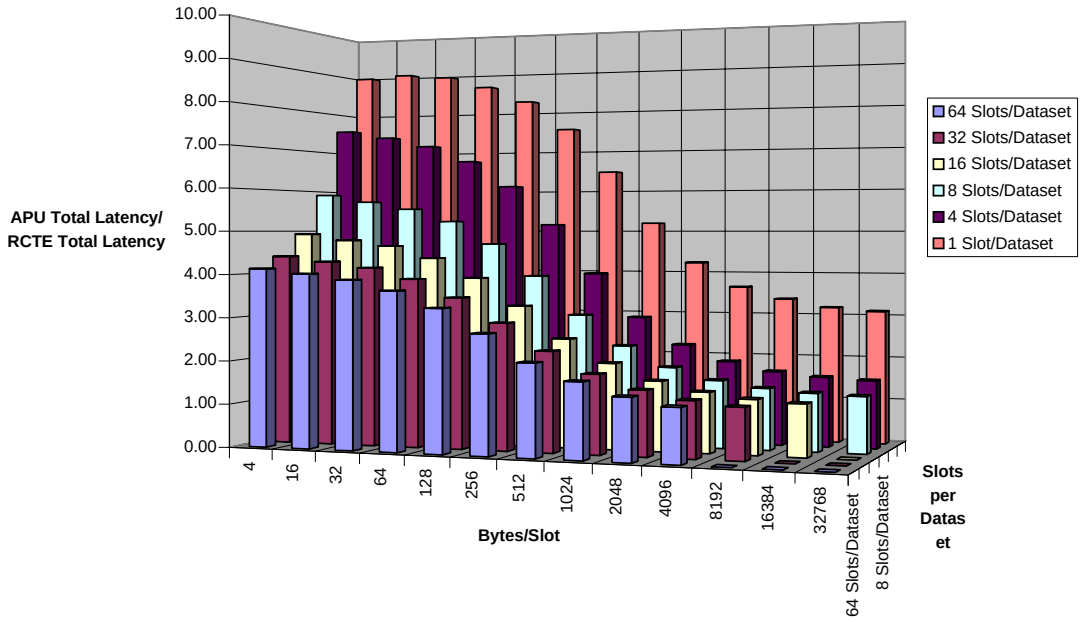



**Figure 53. Ratio of APU Total Latency to RCTE Total Latency**

Note that the multislot data set measurements for both the RCTE and APU assume streamed data, which permits pipelining and therefore more efficient data movement on the LANai. In a real-time system, the application will dictate the data movement, which in all likelihood may not be a streamed set of data slots. A more likely scenario is a scatter/gather operation where multiple sources are feeding a single destination with data slots that make up a single data set. In this case, the realistic comparison between the RCTE and LANai would be represented by the 1 Slot/Dataset set of bars in the back row of the chart above, where messages are unable to be pipelined. For example, in a gather operation using a 1 Kbyte data slot size, the RCTE would be an improvement of more than 5X in data transfer times over the LANai.

Focusing on a simple message passing case clearly contrasts the RCTE and LANai performance. Figure 54. **RCTE/APU Latency and Bandwidth Comparison** shows a comparison on single data slot per data set messages. They demonstrate the fixed overhead latency of the RCTE and the dependence of the APU on data slot size. The bandwidth chart also shows that the RCTE can reach 50% of the network capacity with a data slot size about ¼ the size of one sent/received by the LANai.

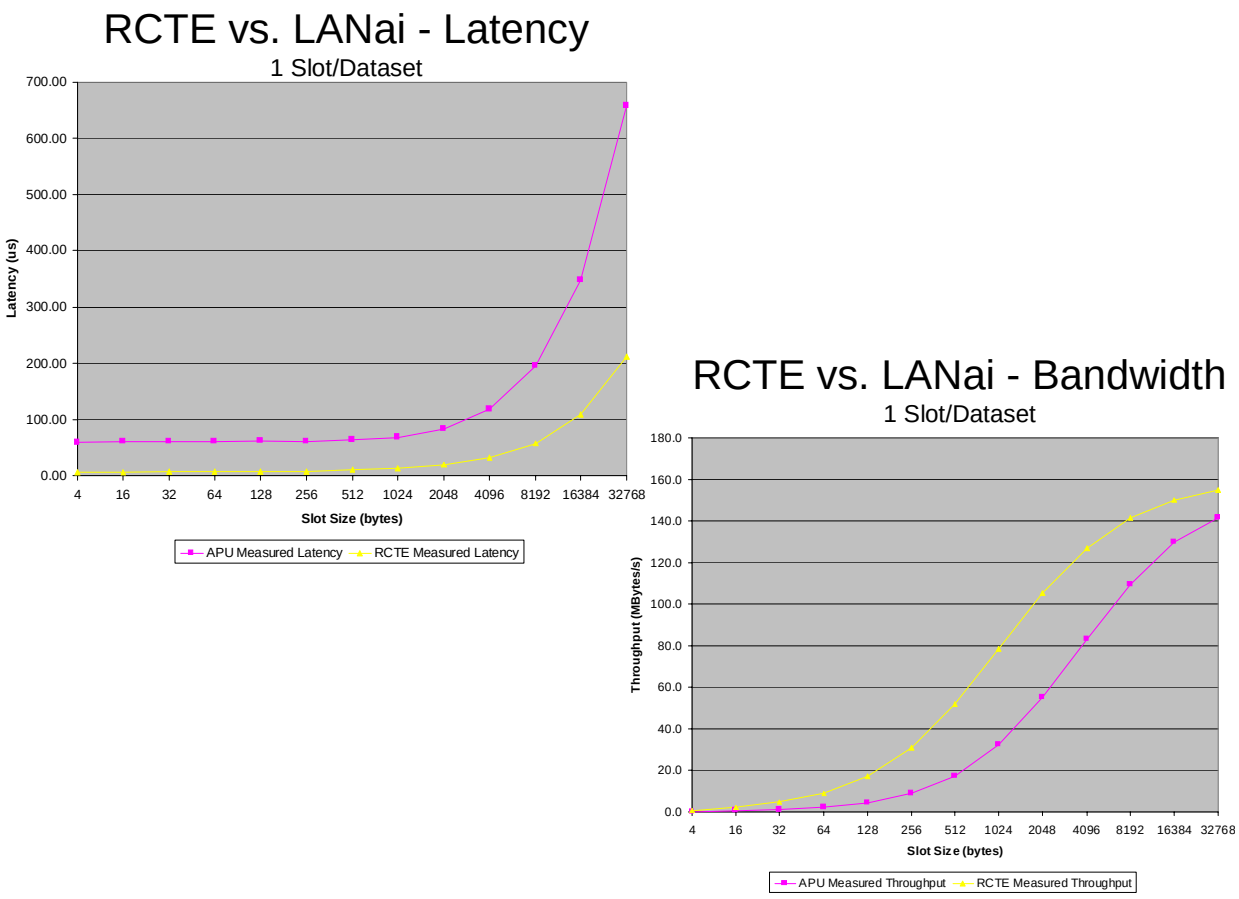



**Figure 54. RCTE/APU Latency and Bandwidth Comparison**

### 4.3.5.2 RCTE Power Performance

The other area of potential performance improvement for the RCTE is in the area of power dissipation. The LANai alone is not sufficient to act as a network controller. The LANai requires a bank of fast SRAM on its local bus for data buffering and processor code. As can be seen in Figure 55, the RCTE network controller components do offer a 25% reduction in power over the LANai-based APU network controller components. To be fair, the APU is a 5V implementation, while the RCTE is a 3.3V implementation. As the table shows, the UUV insertion program 3.3V APU network controller components dissipate 33% less power than the RCTE. This issue must be addressed with additional developments in integrating components of the RCTE.

**RCTE Network Interface**

| Part Num. | Manufacturer | Description | Quantity | Typ Pwr | Max Pwr | Tot Typ | Tot Max |
|---|---|---|---|---|---|---|---|
| FI | Myricom | FIFO Network Interface | 1 | 1.00 | 1.00 | 1.00 | 1.00 |
| MPC860ENZP40 | Motorola | 40 MHz MPC860 Micro | 1 | 0.75 | 1.30 | 0.75 | 1.30 |
| AT29LV040A-20TC | Atmel | 512k x 8 FLASH | 1 | 0.00 | 0.05 | 0.00 | 0.05 |
| MT58LC64K32B3LG-10 | Micron | 64k x 32 SSRAM | 4 | 0.16 | 0.34 | 0.64 | 1.36 |
| OR2T40A-4BC432 | Lucent | 40kgate LV FPGA | 1 | 2.53 | 3.37 | 2.53 | 3.37 |
| IDT72V255L15TF | IDT | 8k x 18 LV sync FIFO | 4 | 0.09 | 0.33 | 0.37 | 1.32 |
| | | | | | Total | 5.29 | 8.40 |

**APU Network Interface**

| Part Num. | Manufacturer | Description | Quantity | Typ Pwr | Max Pwr | Tot Typ | Tot Max |
|---|---|---|---|---|---|---|---|
| MYRICOM | LANai4.1 | Myrinet Interface Chip | 1 | | 3.50 | | 3.50 |
| SAMSUNG | KM64B1003J8 | 256K x 4 SRAM | 8 | | 0.98 | | 7.80 |
| | | | | | Total | | 11.30 |

**3.3V APU Network Interface**

| Part Num. | Manufacturer | Description | Quantity | Typ Pwr | Max Pwr | Tot Typ | Tot Max |
|---|---|---|---|---|---|---|---|
| MYRICOM | LANai4.1 | Myrinet Interface Chip | 1 | | 1.65 | | 1.65 |
| Motorola | MCM6929 | 256K x 4 SRAM | 8 | | 0.50 | | 3.96 |
| | | | | | Total | | 5.61 |

**Figure 55. RCTE/APU Power Comparison**

# 5 Summary

In keeping with the industry's emphasis on COTS products, the EHPSCS program has leveraged core-enabling technologies, commercial, VLSI device technology, and hardware/software design tools and standards to develop a scalable, high performance, real-time multi-level message passing multicomputer testbed. Through the development of a multicomputer tool suite and advanced packaging techniques, the program has provided the system designer with a scalable, high performance, cost-effective solution that supports next-generation application processing requirements and evolving program requirements from prototype to embedded system with environmentally constrained conditions.

The EHPSCS program has been successful in meeting all of its objectives. Specifically, the EHPSCS program:

- Analyzed, developed, demonstrated, and characterized a two-level multicomputer architecture for EHPSCS based on COTS technologies.

- Developed a hardware/software testbed for concept validation. The testbed is supported by a robust development tool suite including a multiprocessor debugger. The testbed and its technologies were made available to the High Performance Computing community to enhance this research technology base.

- Demonstrated the performance and scalability of the testbed with multiple, demanding, real-time algorithms through at least six DoD applications that included, among others, STAP, SAR, EO and IR processing.

- Transitioned the technology developed under the EHPSCS program to at least five DoD insertion programs. The technology was also transferred to the commercial community via the Sanders/CSPI technology licensing agreement. This broadens the technology transfer opportunity while enabling users with COTS support.

- Developed an advanced packaging concept for environmentally constrained DoD applications. The advanced packaging effort successfully reduced size, weight, and power of the architectural realization and explored aspects of reduced MCM cost/risk and reusability.

- Extended the baseline architecture with technology-neutral and hard real-time performance support for a broad range of next-generation insertion opportunities. The RCTE prototype demonstrated an order of magnitude reduction in network communication latency over the EHPSCS baseline.

By using open interface standards and leveraging COTS solutions, the EHPSCS program has demonstrated cost-effective, high-performance, scalable computing solutions for real-time, compute-intensive, next-generation military applications. The packaging technologies demonstrated should also facilitate transferring of the EHPSCS technology for DoD ground, air, and space-based embedded signal processing systems. The EHPSCS technologies already made an impact on the High Performance Computing community. Leading DoD, NASA, and COTS developments that have adopted the EHPSCS technologies include: the Air Force's Improved Space Architecture Concept program, NASA's Remote Exploration and Experimentation program, and CSPI's next-generation DSP product.

# 6 Acronyms

| | |
|---|---|
| API | Application Programmers Interface |
| APU | Arithmetic Processing Unit |
| COTS | Commercial Off The Shelf |
| DARC | Distributed Architecture Resource Controller |
| DARPA | Defense Advanced Research Projects Agency |
| DMA | Digital MicroArchitectures |
| DRAM | Dynamic Random Access Memory |
| DSP | Digital Signal Processor |
| DSQ | Data Synchronization Queue |
| FI | FIFO Interface Myrinet Integrated Circuit |

| | |
|---|---|
| FPGA | Field Programmable Gate Array |
| GE CRD | General Electric Corporate Research and Development |
| GUI | Graphical User Interface |
| HDI | High Density Interconnect |
| KGD | Known Good Die |
| LAN | Local Area Network |
| MCM | MultiChip Module |
| MCP | Myrinet Control Program |
| MI | Myrinet Interface Integrated Circuit |
| MPI | Message Passing Interface |
| MTEM | Myrinet Topology Expansion Module |
| NIC | Network Interface Controller |
| OS | Operating System |
| PMC | PCI Mezzanine Card |
| RCTE | ReConfigurable Transport Engine |
| RISC | Reduced Instruction Set Computer |
| RNI | Resource-Network Interface |
| RTOS | Real-time Operating System |
| SAN | System Area Network |
| SHARC | Analog Devices Super Harvard Architecture Computer DSP |
| SRAM | Static Random Access Memory |
| STAP | Space, Time Adaptive Processing |
| SWAP | Size, weight, and power |
| UUV | Unmanned Underwater Vehicle |
| VLSI | Very Large Scale Integrated |

# 7 Attachment Listing

The following list of documents is attached.

- An Architectural Trade Capability Using the Ptolemy Kernel
- Arithmetic Processing Unit Resource-Network Interface (APU RNI) Component Specification
- Baseline HDI MCM Technology Process Flow
- Hardware Description Document for High Performance Scalable Computing Arithmetic Processing Unit Revision 1, Final Draft
- High Performance Scalable Computing Distributed Architecture Resource Controller Technical Reference
- High-Performance Scalable Computing for Avionics Applications
- High-Performance Scalable Computing for Real-Time Applications
- High Performance Scalable Computing MPI Users Reference Guide
- High Performance Scalable Computing Performance Modeling Using Ptolemy
- High Performance Scalable Computing Primer
- High Performance Scalable Computing Software Users Manual
- HPSC Software Release Notes for EHPSCS Rev. 1.4
- MCM-F Technology Process Flow
- Myrinet-on-VME Protocol Specification Draft Standard, VITA 26-199x, Draft 0.5, 27 January, 1998
- ReConfigurable Transport Engine Technical Specification, Revision 0.64
- SUN Resource Network Interface (SUN_RNI) Component Specification
- Test Procedure, Arithmetic Processing Unit
- Test Procedure, HPSC Chassis
- Test Procedure, Myrinet Topology Expansion Module
- Test Results and Fault Analysis For High Performance Scaleable Computing 3.3V Digital Signal Processor Multi-Chip Module Revision –
- Totalview HPSC Release Note